Rapport de Super-Projet

Grégoire Roussel

25 mars 2019

Présentation du projet

Remerciements

Ce projet constitue mon Super-Projet d'année d'application à l'ENSTA. Je l'ai défini, puis construit, sous la direction de ma tutrice Adina Panchea, et avec l'aide de Sylvie Putot et Éric Goubault (LIX). Je tiens à les remercier chaleureusement pour leur enthousiasme et leurs conseils !

J'ai disposé de la salle d'expérimentation cyber-physique du LIX, au bâtiment Turing (INRIA) pour mes simulations et expériences réelles. Une fois encore, merci au LIX pour leur accueil, et la qualité du matériel mis à disposition!

Portée du rapport

Ce rapport présente les objectifs et limites de ce projet, les choix réalisés, ainsi que les pistes intéressantes à poursuivre par la suite.

Pour faciliter sa compréhension et reprise éventuelle, tout ce qui mérite d'être publié avec le projet est détaillé sur le wiki plutôt qu'ici. Cela comprend:

- la présentation des concepts techniques importants
- la présentation des nodes
- les guides d'installation et d'utilisation
- les résultats obtenus et leur analyse

Une lecture de ce Wiki est ainsi recommandée pour mieux comprendre ce rapport.

Le code source du projet est accessible à : https://github.com/Arpafaucon/sp/

Le wiki est accessible à partir la page du projet : https://github.com/Arpafaucon/sp/wiki/.

Scénario et définition du besoin

La société D possède des vastes entrepôts de stockage. Des fuites sur leur contenu ont encouragé certain importuns à venir explorer la zone, à la recherche d'objets de valeur. Vu la surface à surveiller, les équipes de surveillance existantes sont dépassés.

La société D est prête à investir dans un système d'essaim de drones de surveillance, capables de détecter la présence d'un intrus et d'alerter une équipe de sécurité.

L'objectif du projet est de concevoir et valider expérimentalement un pilote pour un tel système, pour un entrepôt bien défini, d'identifier les points de blocage, et dimensionner l'essaim.

Les besoins identifiés pour le système sont :

- optimiser les rondes des différents agents pour survoler chaque endroit le plus régulièrement possible
- planifier et faire exécuter des trajectoires sans collisions
- détecter des situations de défaillance des drones actifs: batterie, réponse anormale...
- garantir la robustesse du système en remplaçant à chaud les drones défaillants
- fournir un compte rendu visuel de la situation actuelle

Objectifs pédagogiques

L'accent a été mis, non pas sur le développement d'un bloc scientifiquement innovant, mais sur la conception d'un système fonctionnel dans son ensemble, et qui me permette d'approfondir des problèmes récurrents d'un projet informatique complexe:

- paramétrage efficace
- interfaces et communications multi-processus
- utilisation de code externe

Et des thématiques courantes de la programmation robotique :

- localisation
- simulation
- commande et contrôle
- gestion des collisions
- coordination de plusieurs agents
- visualisations

Protocole d'évaluation

Le comportement du système est évalué dans deux phases:

- la **phase statique**: la planification n'est pas active, les drones sont en survol stationnaire. On évalue, lorsqu'une faute est déclarée:
  - le délai avant l'atterissage du drone en disfonctionnement
  - le délai avant la disponibilité d'un drone de remplacement
  - les changements effectués dans l'allocation des missions aux drones pendant cette durée de transition
  - le contrôle et le suivi des manoeuvres de décollage et atterissage
- la **phase dynamique**: la planification est active, les drones recoive des destinations à atteindre (et surveiller). On évalue, lors d'une panne:
  - la vitesse de réadaptation du système à une indisponibilité de drone
  - la vitesse de transfert des missions existantes au nouveau set de drone disponibles
  - l'importance des changements durant cette réallocation
  - la robustesse du système en cas de sur-panne durant la résolution de la première
  - (en permanence) l'évitement des collisions, que ce soit avec des drones en manoeuvres (décollage, atterissage) ou en missions

Réalisation

Matériel et technologies

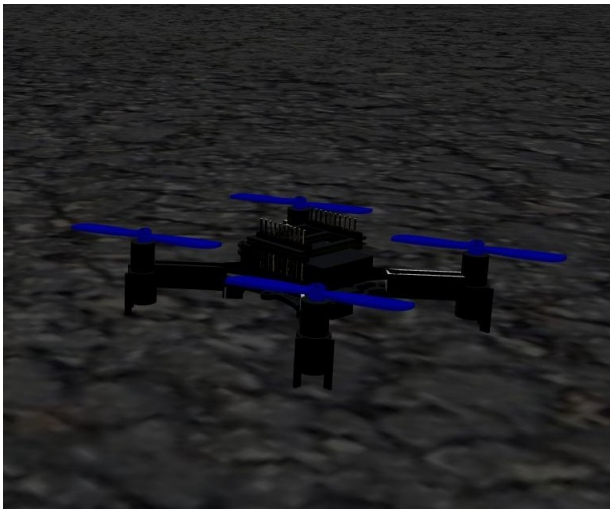
Crazyflie

Dans l'expérience, les drones de surveillances sont des Crazyflie (bitcraze, https://www.bitcraze.io/crazyflie-2/). Ce sont des petits quadri-coptères très légers (environ 40g) contrôlables via radio par un ordinateur central.

Outre leur disponibilité au laboratoire, le choix des Crazyflie se justifie par leur faible coût et leur grande robustesse aux chocs. Par ailleurs, la communauté ROS met à disposition des drivers et un programme de simulation aux **interfaces similaires**. Ainsi, le projet peut indépendamment contrôler des drones simulés ou réels, ou même une combinaison des deux.

La capacité calculatoire des Crazyflie est limitée, mais non-nulle. Ainsi, il est possible de modifier le software originel pour y ajouter des procédures de vérification de la trajectoire, des paramètres de vol, ou prétraiter les données acquise avant envoi. C'est une piste intéressante, qui n'a néanmoins pas été explorée. Elle figure en bonne place des pistes de développement du projet.

drone Crazyflie 2.0



Modèle Gazebo de simulation

Base centrale

La base centrale est le cerveau des opérations. Elle héberge tous les programmes de ce projet, de la planification au contrôle.

Note

La base exécute le méta-OS ROS: http://www.ros.org/ . Ce méta-OS, couramment utilisée lors du prototypage de robots facilite l'interaction entre processus et la conception de blocs fonctionnels modulaires et réutilisables. Elle fournit par ailleurs des outils de développement, de diagnostics et de visualisation qui simplifient la conception d'un système robotique. Dans ce projet, les outils **RVIZ** (outil de visualisation générique) et **Gazebo** (moteur de simulation) sont beaucoup utilisés.

Les deux configurations suivantes ont été testées:

- **PC personel**: Ubuntu 18.04, ROS Melodic, 16Go RAM, 8 CPU, 1To HDD
- **PC de la salle d'expérimentations** Ubuntu 16.04, ROS Kinetic, 64 Go RAM, 40 CPU, 2To SSD

Dans la pratique, l'empreinte mémoire du projet sp est négligeable dans le cas d'une expérience en réel. Pour les simulations, le moteur de rendu physique Gazeo constitue la limite principale en puissance CPU. Il est possible, lorsqu'on réalise une simulation intégrale, d'abaisser la vitesse du temps simulé pour augmenter la précision - une telle pratique ne fonctionnera pas pour un système hybride.

Problèmes rencontrés

Système de localisation

Il est apparu très tôt que le Loco Positioning System proposait une solution clés-en-mains, mais imprécise, au problème de la localisation. Les mesures de localisations obtenues par ce moyen sont relativement justes, mais très peu fidèles: on observe des sauts d'une mesure à l'autre, de l'ordre de la dizaine de centimètre.

Lorsque les drones recoivent des ordes en position, ces sauts induisent une forte instabilité, qui peut aller jusque causer la chute du drone. Cela nuit drastiquement à la qualité des expériences avec des drones réels.

Pour résoudre ce problème, j'ai adapté mon contrôle en position de manière à augmenter la tolérance et limiter les micro-ajustements, qui sont les opérations les plus risquées. Cela améliore la situation légerement, d'autres pistes sont évoquées en fin de rapport.

Fiabilité des communications

Il est apparu, plus tard dans l'année cette fois-ci, que certains canaux de communication n'étaient pas assez fiables pour être utilisés tels quels. En voici deux exemples:

Publication de positions invalides: lorsque la connection avec le drone est mauvaise, le driver CRTP-ROS ne publie pas toujours la position réelle du drone. Lorsque les données n'ont pas pu être récupérées, une trame nulle est publiée sur le topic de position. Avant sa détection, ce comportement plongeait le système dans un état incohérent, certains drones en vols étant détectés sous le sol (et en collision si cela arrivait à deux drones en même temps). Un filtrage a résolu ce problème, au prix d'une perte de fréquence des messages de position.

Conflit d'ordres: Lorsqu'un drone est sorti du pool actif, il reçoit un ordre d'atterissage. Le fonctionnement interne de ROS fait qu'il peut arriver qu'un ordre de déplacement émis avant la sortie du pool soit reçu aprés le début de la phase d'atterissage. Cette phase est annulée, et le drone suit les indications de l'ordre retardataire. Pour résoudre ce problème, la gestion des changements d'états a été rendue plus robustes. Lors d'un atterissage, par exemple, l'ordre d'atterir est émis périodiquement, jusqu'à ce que le drone

atteigne une altitude-plancher.

Propagation des résultats

La propagation des ordres dans le système fonctionne en cascade : les ordres de l'admiral déclenchent le travail du capitaine, puis celui du second. Mais cette propagation prend du temps (de l'ordre de quelques secondes, la plupart dues à la planification de trajectoire).

Il peut arriver que l'allocation change pendant cette propagation. Dans ces cas, le capitain risque alors de demander la position d'un drone actif qui n'existe plus, ou au contraire émettre des ordres qui ne prennent pas en compte un drone tout juste devenu actif. Ces situations causent au mieux un perte d'efficacité, au pire une erreur dans le programme.

Ainsi, un défaut dans la propagation des changements d'allocation dans `lookout` causait une mauvaise association des positions aux drones actifs, et à des collisions systématiques.

Pour y remédier, il est nécessaire de rester prudent avec la temporalité, et de s'assurer au début de chaque calcul qu'il est encore pertinent. De nombreuses vérifications aux étapes-clés garantissent la stabilité du programme, et donc la robustesse du système.

Contributions à sim_cf

Outre la production de la pile logicielle de ce projet, j'ai pris beaucoup d'intérêt à collaborer avec l'auteur d'un des *stacks* impliqués (sim_cf). De simple utilisateur au début, j'ai eu l'occasion de modifier son code et l'adapter à mes besoin, et faire remonter ensuite les améliorations aux autres utilisateurs. En particulier:

- le port du plugin de simulation de Gazebo 7 à Gazebo 9
- la réorganisation de la structure des modules python de `crazyflie_gazebo` pour tirer parti des fonctionnalités de mise en partage de ROS
- l'écriture / complétion de scripts, utilitaires pour faciliter l'utilisation du projet.

Les contributions et discussions sont accessibles publiquement sur la page du projet, au rubriques `issues` (https://github.com/wuwushrek/sim_cf/issues/) et `Pull requests` (https://github.com/wuwushrek/sim_cf/pulls).

Bilan du projet

On peut identifier les aspects réussis, et à perfectionner:

Aspects positifs

Au terme du projet, le système a atteint un fonctionnement cohérent, et robuste aux fautes de drones en simulation. L'élaboration et le suivi des ordres sont fonctionnels, et la méthode d'évitement des collisions donne des résultats satisfaisants en terme de sécurité et d'efficacité (pas de blocages inutiles).

Sur le plan pédagogique, ce projet m'a permis d'explorer plusieurs composantes du travail de développement en robotique:

- l'architecture d'un système et la communication entre ses composants internes
- l'interfaçage avec des projets indépendants
- la contribution aux outils open-sources utilisés
- la simulation (avec Gazebo) : écriture d'un plugin et d'un modèle URDF
- la mise au point d'une stratégie pour répondre optimalement à un besoin (ici, de couverture)
- la planification de trajectoire (multi-agents)
- le contrôle, et l'évitement de collisions
- la gestion de fautes, et de l'évolution du système au cours du temps
- la gestion des délais et débits de transmission de l'information
- la compréhension et la visualisation du fonctionnement (ici, avec RVIZ)

Ces aspects sont trop nombreux pour que je puisse les aborder en profondeur dans un projet de 6 mois. Ainsi, je ne prétend pas avoir atteint des solutions optimales, et utilisables industriellement.

J'ai en revanche pris le temps, pour chacun de ces points, de construire une solution primitive et fonctionnelle, que j'ai fait ensuite évoluer selon mes besoins.

Aspects à poursuivre

Réalisme et complexité du modèle

À ce stade, la modélisation de la surveillance de l'entrepôt présente de nombreuses limites:

- Une incohérence majeure : la zone considérée comme "observée" par un drone est celle située en-dessous de sa position au

moment du début du calcul des cibles par l'amiral. Les zones survolées en chemin ne sont pas considérées.
- La durée de survol statique de la zone "observée" ne rentre pas en ligne de compte pour déterminer la qualité de cette observation. Une observation instantanée n'est pas suffisante pour effacer tout soupçon
- Il est inutile de "jeter un oeil" dans une zone : un voleur peut très bien se cacher dans un coin en entendant un drone arriver, et reprendre son méfait ensuite. Un modèle de diffusion des incertitudes (une zone très bien observée à côté d'une inconnue est moins intéressant que l'ensemble surveillé moyennement) permettrait de travailler sur des stratégies de surveillance plus complètes.
- Toutes les zones n'ont pas le même risque - et donc pas la même fréquence optimale de survol
- L'intérêt d'une zone ne croît pas nécessairement linéairement

Ces observations plaident en faveur de la redéfinition du modèle de la zone à surveiller, et des indicateurs de performance à optimiser. Cela contribuerait à faire évoluer des stratégies de surveillance plus complexes, et plus efficaces.

Précision de la localisation et contrôle

Comme on peut le lire plus haut, et le constater par les vidéos de démonstration, le système de localisation est **le point noir du système actuel**.

Quelques pistes pour résoudre ce problème seraient:

- le changement de système de localisation: l'installation d'un système OptiTrack (système optique, précision de l'ordre du mm) à l'ENSTA permettra de juger des améliorations à espérer
- le passage à un contrôle plus bas niveau sur les vitesse: parfaitement faisable, mais plus chronophage, car il faut déterminer les paramètres dynamiques appropriés
- la modification du code embarqué du Crazyflie pour "amollir" sa réponse aux sautes de position et augmenter sa stabilité (changement des coefficients du PID)

Collisions et obstacles

La procédure de détection et de résolution des collisions est loin d'être parfaite. Il existe encore plusieurs scénarios de bloquage, et pire, de non-protection: par exemple quand deux drones arrivent face-à-face. En l'absence de capteurs de proximité, cette méthode doit gagner en fiabilité et permettre une meilleure anticipation.

Des techniques de navigation fondées sur le flux optique ("Navigation des drones par flux optique", S. Doncieux et A. Angeli) semblent prometteuses pour résoudre les conflits. L'article "Trajectory Planning for Quadrotor Swarms" (W. Hönig et al.) fournit par ailleurs une approche calculatoire, mais garantie, pour planifier des trajectoires spatio-temporelles exemptes de collisions.

Par ailleurs, aucune vérification n'est faite au bas-niveau sur la présence d'obstacles fixes. En l'absence de capteurs, cela revient à vérifier sur la carte que le drone ne se dirige pas vers des cases occupées.

Vu la précision du système de localisation, une telle vérification manque encore de pertinence, et n'a pas été jugée prioritaire à implémenter. Elle est néanmoins nécessaire, car les trajectoires issues d'un RRT peuvent accidentellement "rogner" sur un coin d'obstacle.

Protection des drones durant les manoeuvres: pour l'instant, le système de prévention de collisions ne surveille que les drones en altitude de vol et disponibles au commandes. Lorsqu'un drone décolle ou atterit, il existe un bref instant ou il est invisible pour les collisions - mais à l'altitude de travail.

Une extension du processus de contrôle corrigerait ce dernier point.

Détection interne et externe de fautes

Détection automatique interne de fautes: pour ajouter à la pertinence du système, il serait intéressant que le drone embarque un module d'autodiagnostic sur sa position, sa réponse dynamique, sa batterie... Le système central pourrait alors recevoir des rapports de pannes et poser les drones en conséquence.

Intégration de code dans les drones et décentralisation

En complément du point précédent, il peut être très intéressant de modifier le firmware des drones pour y ajouter des tâches de première nécessité, et ne souffrant pas les délais et imperfections des communications avec la base.

Ça serait en particulier le cas des procédures de détection et de résolution de collisions, ainsi que des routines de décollages/d'atterissage. Cela garantirait un fonctionnement, dégradé mais fiable, en cas de panne ou de déconnexion de la base centrale.

Adaptation des périodes d'observation

Pour finir, il faut remarquer que le système fonctionne encore sur la base de périodes d'observations fixes. Il serait pertinent, et plus

efficace, de permettre la variation de la fréquence de calcul de positions:

- si l'on est dans une phase de changement d'allocation, pour suivre les changements de capacité opérationnelles et en tirer parti au mieux
- si l'on constate que les drones arrivent systématiquement en avance sur leurs objectifs.
- si un évenement suspect est identifié, afin de maintenir une qualité d'information suffisante le temps que les équipes d'intervention arrivent.