Surveillance du golfe de Gascogne

15 février 2016

Introduction

Le but de ce projet est de détecter des intrus dans une zones maritime donnée. Ici, nous allons nous intéressé au golfe de Gascogne. Nous allons donc utiliser un groupe de robots équipés de GPS et nous allons implémenter des algorithme et un stratégie pour assurer la sureté d'une zone.

Nous allons avoir plusieurs hypothèses de départ :
— Nous supposeront que l'étude peut être simuler dans un monde en deux dimensions.
— Les intrus auront une vitesse constante tout au long de la simulation.
— Tous les robots pourront voir autour d'eux sur une distance d_i et détecter si un intrus intrus est présent sur cette zone.
— Nous définissons une zone sure comme un ensemble de point pour lequel un intrus ne peut se trouver.

Nous allons donc présenter au final deux livrables :
— Une simulation théorique fait en grande partie en langage python qui permettra de déterminer la pertinence de nos algorithmes.
— Un simulation sur des robots char disponible au club robotique de l'ENSTA Bretagne.

Chapitre 1

Quelles stratégies ont été retenues

Expliquer comment nous allons répondre aux exigences
1) description de la trajectoire retenue 2) Méthode retenues (OpenCV / théorie des intervalles)

Chapitre 2

Mise en œuvre des solutions

Trajectoire elliptique

Répartition des bateaux Dans un premier temps, nous avions penser à répartir les bateaux avec un espacement régulier selon l'abscisse curviligne de l'ellipse. Cette méthode étant finalement très difficile à mettre en place d'un point de vue mathématique et algorithmique, cette solution a été abandonnée. Il aurait fallu, pour se faire, approximer les intégrales de Wallis par un développement limité assez complexe, ce qui implique un temps de programmation important et un temps de calcul beaucoup trop long pour une simple étude de faisabilité. C'est pour cela que nous avons décidé de nous orienter vers un retard angulaire sur chacun des robots. Ainsi, chaque robot suit le robot précédent avec un retard angulaire fixe dépendant du nombre de robots parcourant l'ellipse. Cela implique que les robots vont être d'autant plus proches les uns des uns qu'il sont près des apogées de l'ellipse, et d'autant plus éloignés aux périgées. Néanmoins, malgré la non-linéarité du placement des robots, nous n'avons pas de rupture de la zone de détection. En effet, comme les robots vont plus vite aux périgées, la trace qu'ils "laissent" est plus grande, et inversement aux apogées. De ce fait, on peut garantir la sécurité de la zone à surveiller, même si les robots ne sont pas régulièrement espacés.

2.1 Évolution de la consigne au cours du temps

Alice / Eric / Chuch

2.2 Régulation de la meute de robot

Alice / Eric / Chuch ou Totoboule / Sylvain selon la solution retenue

2.3 Description du fonctionnement du simulateur

Totoboule / Sylvain + Image à faire

2.4 Estimation de la zone de sureté grâce à la théorie des intervalles

Alice / Eric / Chuch / Elouan ... au choix mais je pense que Elouan est le plus connaisseur ;)

2.5 Prise en compte du passé grâce à OpenCV

Maël / Khadim / David ... A vous de choisir vos sous-sections

Pavage des images

Érosion des zones de sureté

Chapitre 3

Réalisation de test sur robot char

Benoit / Maxime / Pierre ... A vous de choisir vos sections

Pour la réalisation d'une simulation sur des robots, nous avons du implémenter un architecture afin de permettre à nos programme de gérer d'une part nos robots et d'autre part de voir si le résultat théorique correspond aux résultats expérimentaux.

Nous aboutissons au final à l'architecture suivante :

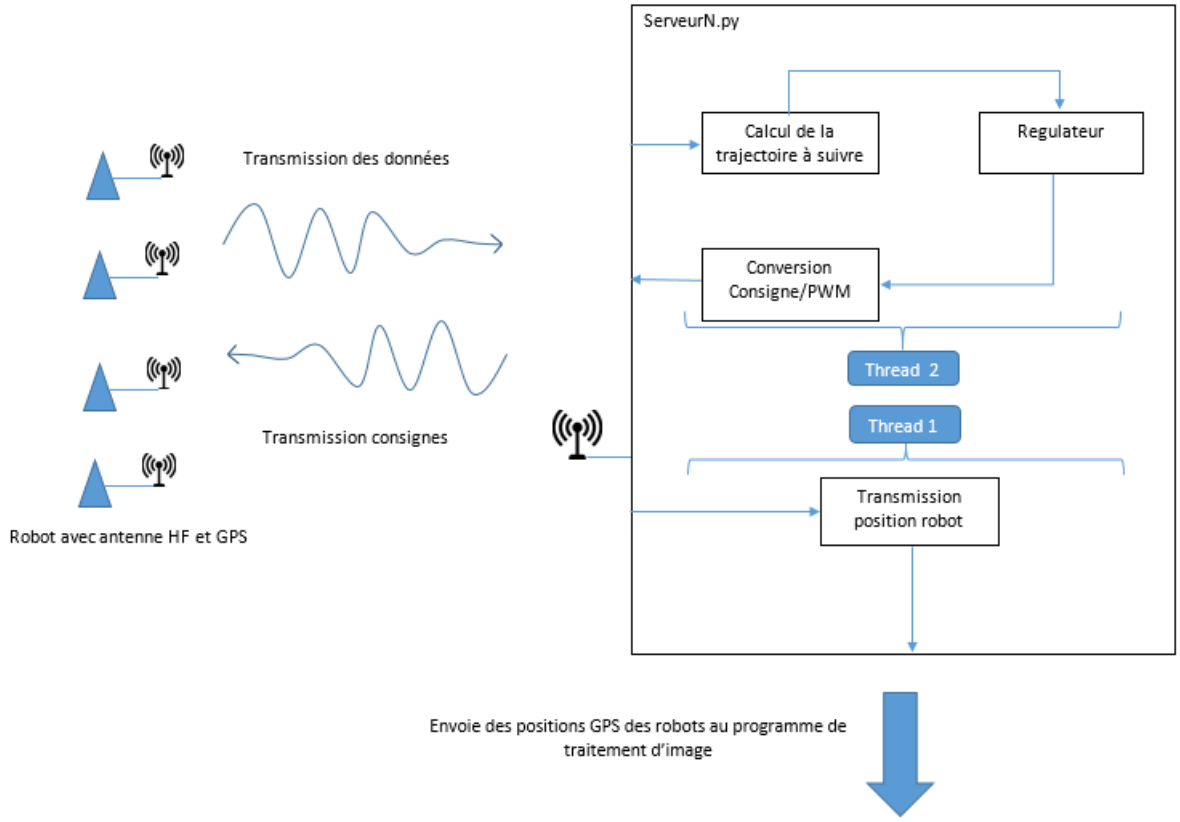

FIGURE 3.1: Architecture mise en place pour la simulation sur robot char.

3.1 Matériel

3.1.1 GPS

ici la description des GPS utilisés

3.1.2 Robot

ici la description des robots et des récepteurs

3.2 Analyse des trames GPS

Pierre JACQUOT Afin de récupérer les coordonnées GPS des robots nous avons décidé d'utiliser la tram *GPRMC* envoyée par les émetteurs GPS. L'exemple ci-dessous montre l'aspect d'une trame :

081836	A	3751.65	S	14507.36	E	000.0	360.0	130998,01	011.3	E*62
heure UTC	Status des Donneés	Latitude	N ou S	Longitude	E ou W	Vitesse en nœuds	Cap	Date UT	Déviation Magnétique	E ou W et Check-sum

Ici les données importantes sont la longitude, la latitude le temps et la vitesse au niveau du sol. Afin de les récupérer, j'ai utiliser le package python pynmea, qui permet de facilement récupérer et parser une trame GPS. La latitude et la longitude récupérées ont ici une structure particulière qu'il est nécessaire de modifier afin d'avoir des données compréhensible et facilement exploitable. L'exemple ci dessous montre la structure de la longitude et de la latitude

— Longitude : 12311.12,W Longitude 123 degrée. 11.12 min Ouest
— Latitude : 4916.45,N Latitude 49 degrée 16.45 min Nord

Les directions cardinales associées la latitude (Nord et Sud) et à la longitude (Est et Ouest) permettent de savoir respectivement de quel côte du méridien de Greenwich ou de l'équateur nous nous trouvons. En l'occurrence, nous aurons à Brest une latitude "négative" et une longitude positive.
D'un point de vue programmation, il a donc était nécessaire de convertir la longitude et la latitude en degré et de prendre en compte les directions cardinales afin de rendre négative ou non la longitude et latitude.
En plus de la latitude et de la longitude, notre programme permet aussi de récupérer les coordonnées UTM (Universal Transverse Mercator). Ces coordonnées permettent de représenter la position des robots dans un repère local et peuvent être plus facile à manipuler que des longitudes et latitudes car sont simplement sous la forme de nombres décimaux. Le système UTM se base sur une division du monde en secteur, et permet ainsi d'obtenir des coordonnées locales pour chaque secteur. Les secteurs découpants la France sont décrits dans la figure ci-dessous :

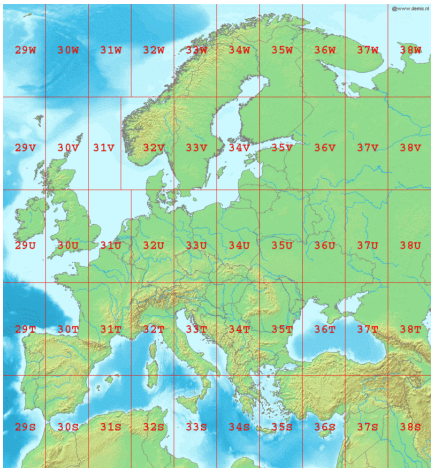

FIGURE 3.2: Secteur UTM

On constate à l'aide de la figure précédente que nous nous trouvons dans le secteur 30. Cependant afin d'effectuer ses projections par secteur, il faut aussi choisir un système géodésique représentant la Terre. Nous avons choisi le système le plus courant pour ce projet, soit le système WGS-84.

3.3 Structure du code

Quatre scripts python : GPS2.py, commande.py, testserial.py et ServeurN.py

3.4 Résultats