ExploBot
Dossier de Conception

Version 1

Auteur	Équipe ExploBot	simon.fedeli@reseau.eseo.fr jeaneudes.laguerie@reseau.eseo.fr antoine.dehoux@reseau.eseo.fr fatoumata.traore@reseau.eseo.fr thomas.rocher@reseau.eseo.fr
Collaborateur	Pierre MADDONINI Bastien ROCOURT Thomas CHENU Mario RAJAOARISOA	
Encadrant	Matthias BRUN	

Vu d'ensemble

Ce document de conception concerne la conception générale du prototype de ExploBot. Cette partie présente l'Architecture Candidate et donne les grands principes de fonctionnement de notre projet. Elle détaille ensuite chaque composant du système, en présentant pour chacun leur description structurelle.

1. Introduction

1.1. Objet

Ce dossier de conception a pour objectif de rassembler toute la conception du logiciel ExploBot. Il permettra à l'équipe PFE de développer le logiciel ainsi que d'élaborer des tests.
Les éléments de conception présentés dans ce document ont été déterminés à la suite de l'étude du dossier de spécification Spécifications ExploBot.
Ce dossier de spécification s'inspire de la norme [IEEE/ISO/IEC 29148-2018]. Il utilise des schémas et illustrations respectant la norme UML en version 2.5.1 [UML 2.5.1_ 2017].

1.2. Portée

Ce document décrit les éléments de conception du Système à l'Etude (SaE).
Il est destiné :

- À l'équipe de développement C et Python afin de préciser l'implémentation des objets constituants le SàE.
- Aux testeurs, afin qu'ils puissent élaborer les tests adéquats vérifiant la philosophie de conception adoptée par l'équipe.
- Au client pour que le cadre du projet et la direction prise par l'équipe soient clairs et dans la continuité des spécifications.

1.3. Définition, acronymes et abréviations

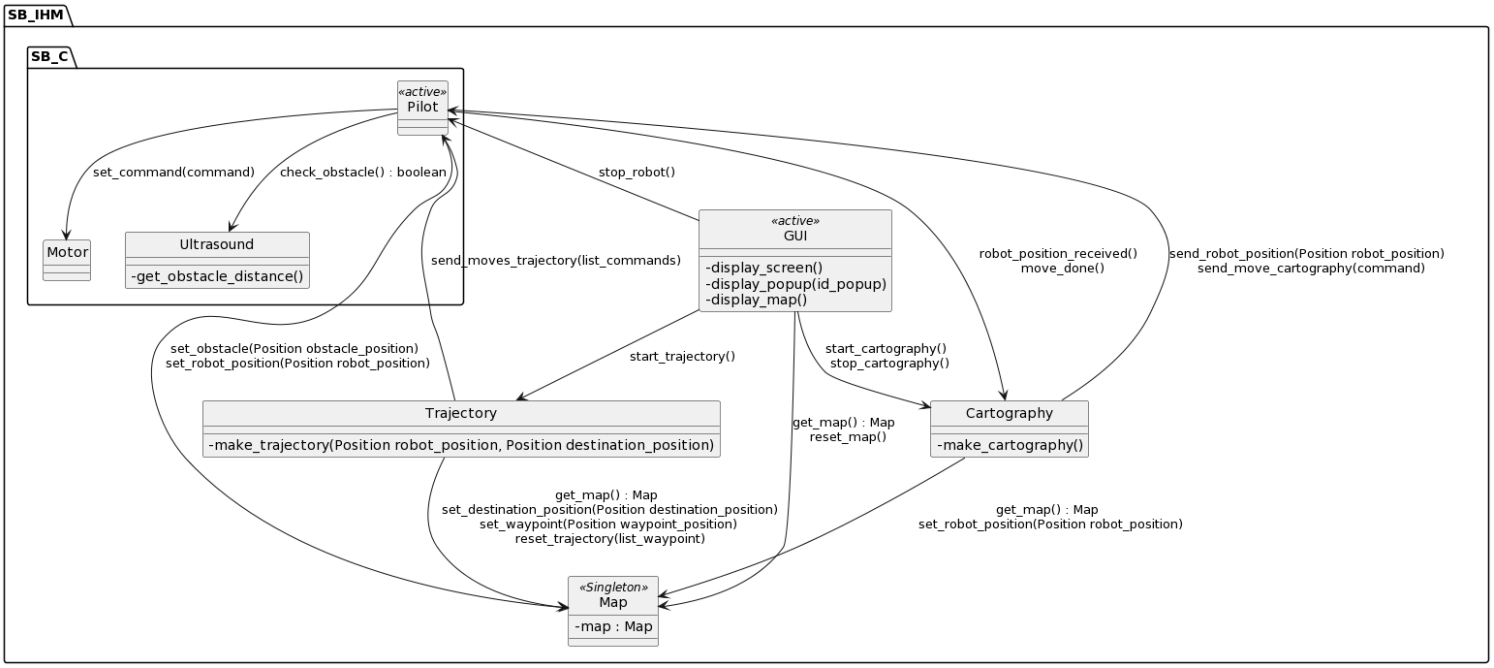
Les abréviations utilisées dans le présent document sont répertoriées et expliquées dans le tableau présenté ci-dessous. Les termes utiles pour interpréter correctement ce dossier de spécifications sont définis dans le dictionnaire de domaine présent dans ce dossier dans la partie 3.3.

Acronymes, abréviations	Définition
CU	Cas d'Utilisation
ID	Identifiant
IEEE (Institute of Electrical and Electronics Engineers)	Association professionnelle internationale définissant entre autres des normes dans le domaine informatique et électronique.
IHM (Interface Homme Machine)	Moyens permettant aux utilisateurs de ExploBot d'interagir avec ExploBot
N.A	Non Applicable
SàE (Système à l'étude)	Il s'agit de l'ensemble des logiciels ExploBot_CONTROLEUR et Cute.
Carto	ExploBot_Controleur, contrôleur utilisé pour les robots du SàE
Cute	ExploBot_IHM. Interface Homme-Machine utilisée pour le SàE
UML (Unified Modeling Language)	Notation graphique normalisée, définie par l'OMG et utilisée en génie logiciel.

2. Conception générale

2.1. Architecture candidate

Dans le diagramme suivant, nous présentons l'Architecture Candidate de notre SàE. Les différents objets y sont présents ainsi que les interactions qu'ils entretiennent. On peut également observer les objets actifs ainsi que l'organisation du logiciel.



Architecture Candidate de Explobot représentée par un diagramme de communication

Les objets suivants forment l'application Cute, destinée à être exécutée sur un PC ou une tablette Android :

- GUI : gestion de l'affichage de l'écran et de ses composants (boutons et carte).
- Cartography : gestion de la cartographie.
- Trajectory : calcul de la trajectoire optimisée vers le point de destination.
- Map : contient la carte.

Les objets suivants forment l'application Carto, destinée à être exécutée sur une raspberry 3B+ dans le cadre de ce prototype :

- Motor : gestion des déplacements du robot.
- Ultrasound : gestion du capteur ultrason, donc la détection d'obstacle.
- Pilot : gestion de Motor et Ultrasound, donc le pilotage global du robot.

2.2. Grands principes de fonctionnement

Cette partie présente le fonctionnement général de Explobot. Cela est représenté sous forme de diagrammes de séquence. Pour chaque diagramme de séquence, un cas nominal est racontée, les erreurs ne sont pas gérées dans ce type de diagramme.
Voici la liste des diagrammes de séquences présentés dans la suite de ce document :

- Déplacer le robot en toute autonomie
- Le robot cartographie la zone où il se trouve
- Optimiser la trajectoire
- Réinitialiser la cartographie

2.2.1. Scénario général

Ce diagramme représente le scénario nominal du CU stratégique du dossier de spécifications. L'utilisateur démarre Cute afin de pouvoir commencer la cartographie de la zone par le robot ainsi que le déplacement autonome du robot. L'utilisateur peut également quitter le logiciel.

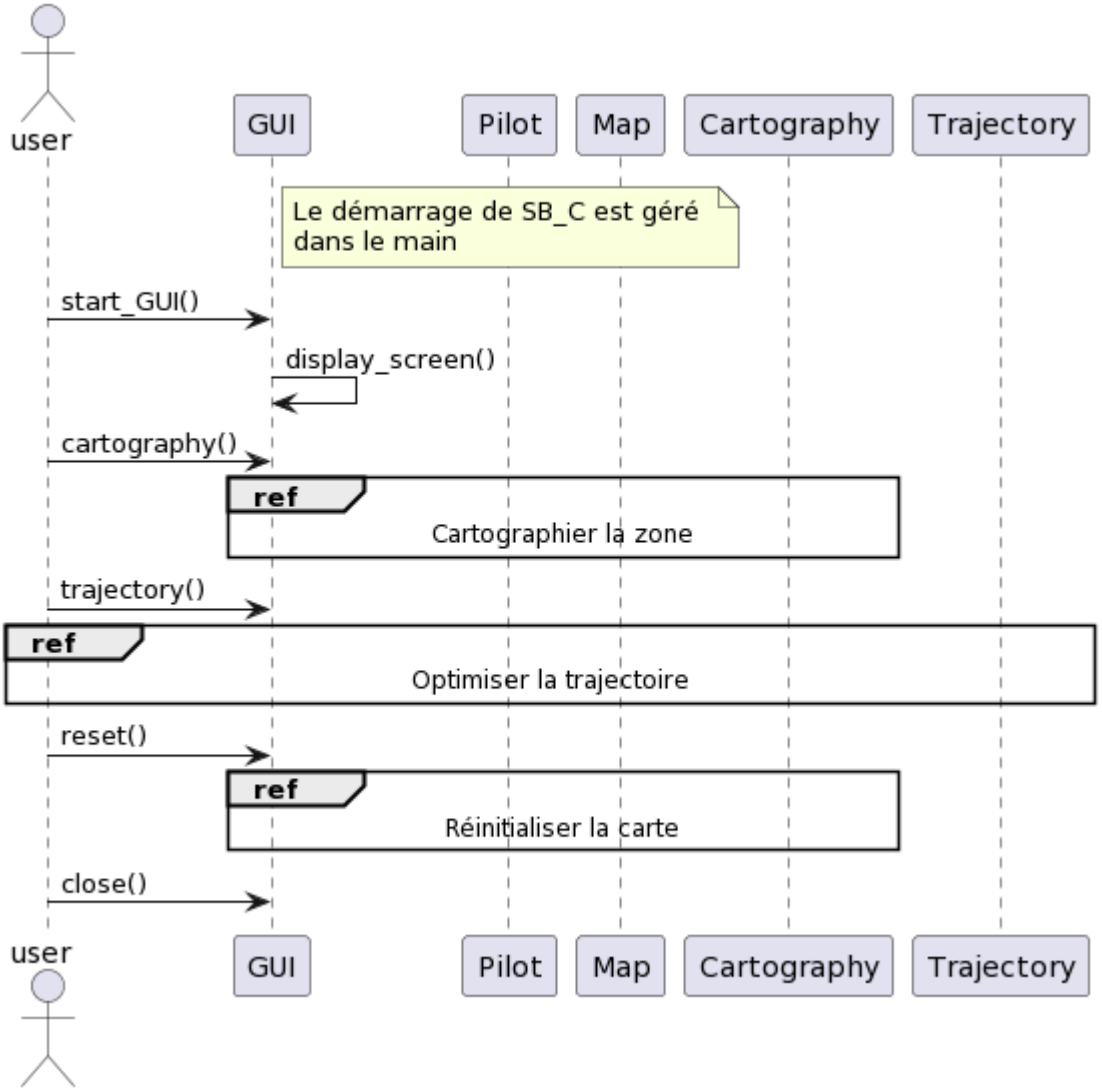


Diagramme de séquence du scénario général

2.2.2. Scénario de cartographie

Ce diagramme représente le scénario du CU « Cartographier la zone » du dossier de spécifications. Le robot réalise la cartographie de la zone dans laquelle il se trouve. L'utilisateur peut consulter la carte via l'IHM.

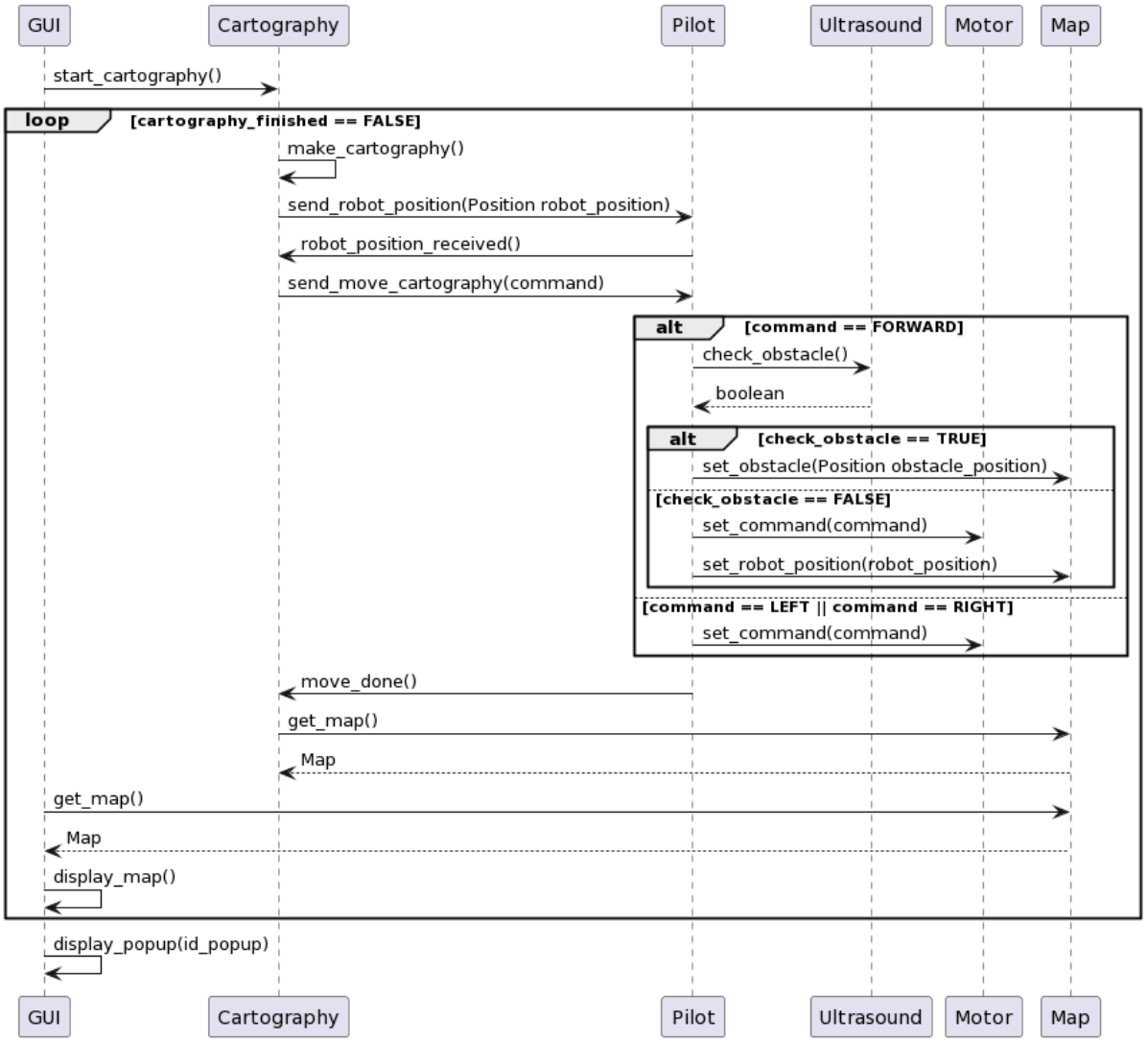


Diagramme de séquence du scénario de cartographie

2.2.3 Scénario de trajectoire

Ce diagramme représente le scénario du CU « Optimiser la trajectoire » du dossier de spécifications. Une fois la cartographie réalisée, l'utilisateur indique au robot un point de destination sur la carte, celui-ci définit la trajectoire optimale pour s'y rendre et l'effectue.

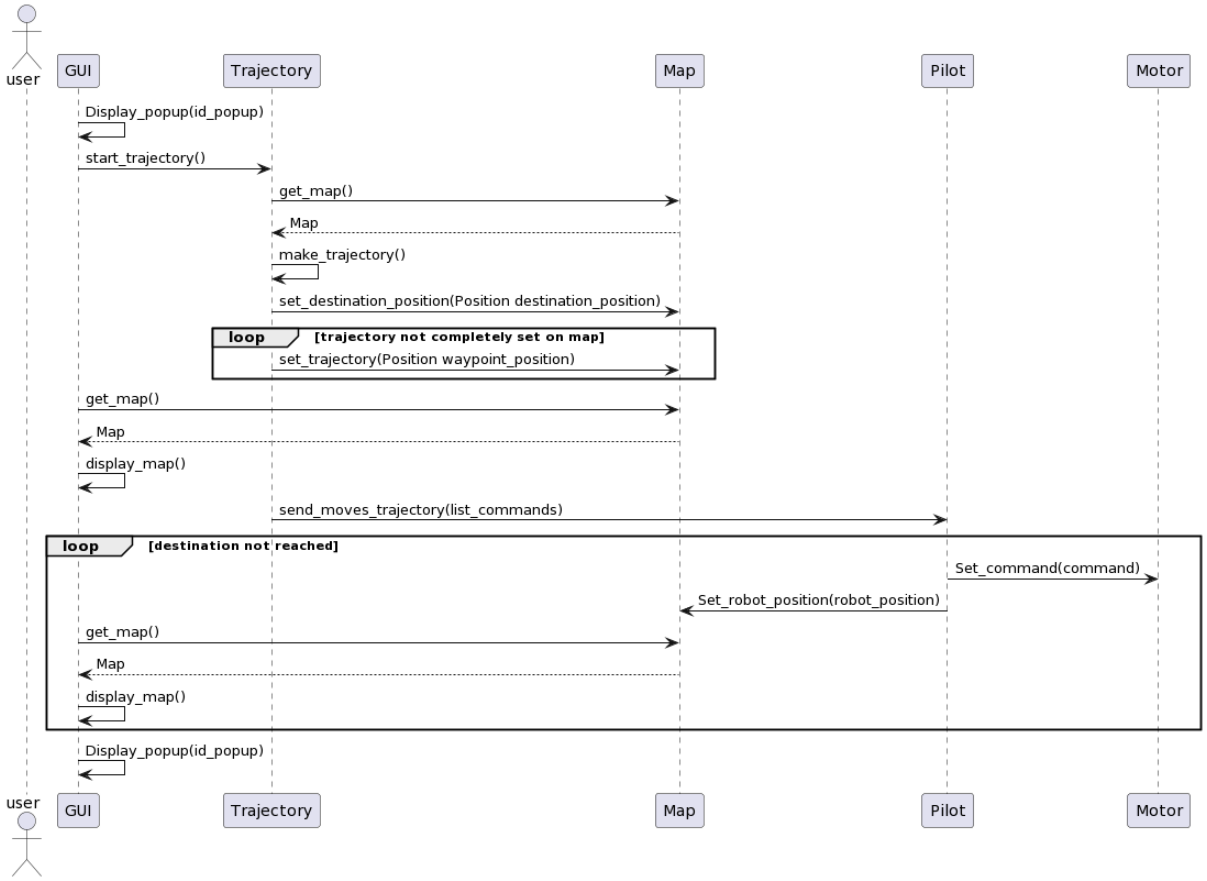


Diagramme de séquence du scénario de trajectoire

2.2.4 Scénario de réinitialisation

Ce diagramme représente le scénario du CU « Réinitialiser la carte » du dossier de spécifications. L'utilisateur initialise Cute, l'espace mémoire de la carte est vidé et la carte affichée sur Cute est supprimée.

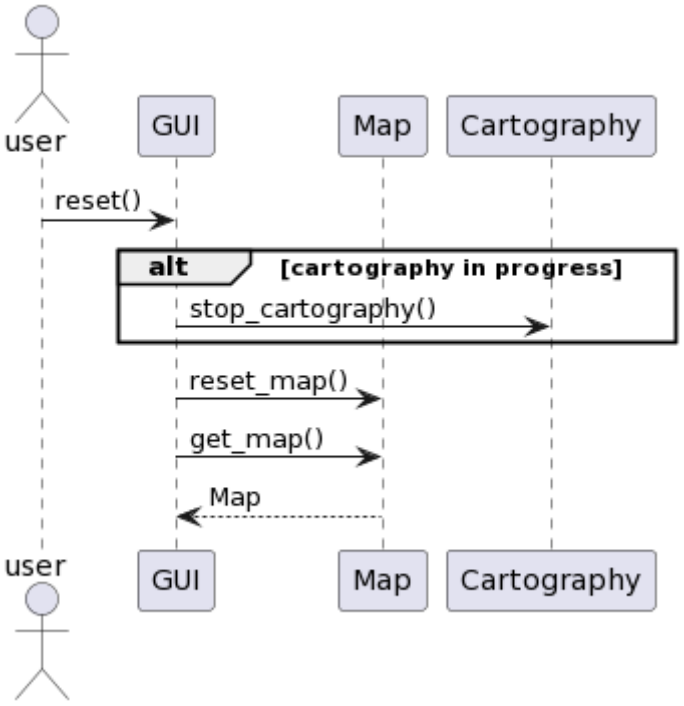


Diagramme de séquence du scénario de réinitialisation

2.3 Description des composants

2.3.1. [IHM] La classe GUI

Diagramme de classe de la classe GUI

2.3.1.1. Philosophie de conception

La classe GUI représente la classe principale de Cute. Elle s'occupe de l'interface utilisateur entre Utilisateur et Cute et permet également d'afficher les vues.

2.3.1.2. Description structurelle

Attributs

N.A.

Services offerts

- **(-)display_screen() : void :** Méthode privée affichant l'IHM.
- **(-)display_popup(id_popup) : void :** Méthode privée affichant une popup à l'écran dont le contenu dépend de id_popup.
- **(-)update_map() : void** : Méthode privée mettant à jour la carte sur l'IHM

2.3.2. [IHM] La classe Cartography

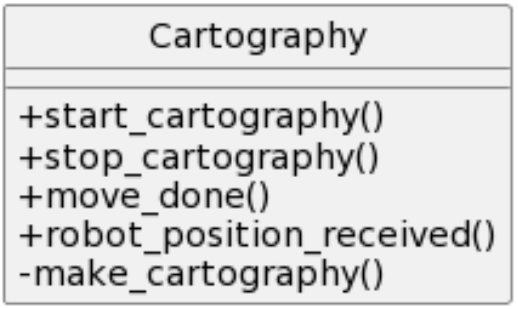


Diagramme de classe de la classe Cartography

2.3.2.1. Philosophie de conception

La classe Cartography permet de cartographier une zone pré-délimitée. Il exécutera les calculs d'esquive d'obstacle tout en suivant un pattern de déplacement.

2.3.2.2. Description structurelle

Attributs

N.A

Services offerts :

- **(+)start_cartography() : void :** Méthode publique permettant de commencer la cartographie de la zone.
- **(+)stop_cartography() : boolean :** Méthode publique permettant d'arrêter la cartographie de la zone.
- **(+)move_done() : void :** Méthode publique permettant de confirmer que le robot a bien effectué une commande de déplacement envoyée pour la cartographie.
- **(+)robot_position_received() : void :** Méthode publique permettant de confirmer que la position du robot a bien été envoyée à Pilot, et qu'il peut commencer à recevoir des commandes de déplacements.
- **(-)make_cartography() : void :** Méthode privée permettant de d'exécuter l'algorithme de calcul de cartographie de la zone.

Algorithme de la méthode make_cartography() :

Choix de l'algorithme :

L'algorithme utilisé a été entièrement fabriqué par l'équipe. Il en a été décidé ainsi suite aux problèmes techniques du robot et de la méthode de stockage de données pour la carte. Il est cependant simple d'utilisation et de compréhension.

Méthode de stockage :

Les données de la carte sont stockées dans une matrice. C'est-à-dire que la représentation se fera sur une grille remplie de nombres. Le robot fait la taille d'une case dans cette matrice

0 - Une case blanche est considérée comme vide.
1 - Une case bleue est considérée comme un endroit où le robot est déjà passé.
2 - Une case rouge est considérée comme un obstacle.

Méthode de déplacement :

Le robot se déplace en serpentin en partant du coin le plus en haut à gauche de la zone. Il se déplace de manière à cartographier toute la zone ligne par ligne, en commençant par la ligne la plus en haut, puis s'occupe de la ligne en dessous à répétition jusqu'à ce qu'il ait tout cartographié.

Il se déplace ainsi :
- Cartographie la ligne la plus haute de gauche à droite.
- Puis descend d'une case et s'occupe de la ligne de droite à gauche.
- Puis descend les lignes en suivant ce même pattern.

Les déplacements du robot sont effectués via une série d'appel à la fonction :
send_move_cartography(command)

Méthode de contournement :

Le contournement d'un obstacle s'effectue en suivant le contour de l'obstacle, en empruntant le chemin le plus court et en privilégiant les zones déjà cartographiées. Étant donné que le robot ne peut pas accéder à la case directement devant lui, son objectif sera de se déplacer vers la case suivante de son itinéraire.
Si le robot détecte un autre obstacle derrière cette case, il cherchera à atteindre à nouveau la case suivante. Ce processus se répète jusqu'à ce qu'il atteigne une case sans obstacle. Ensuite, il reprend son déplacement par défaut.
L'algorithme utilisé est A* comme pour la méthode Trajectoire défini ci-dessous.

2.3.3. [IHM] La classe Trajectory

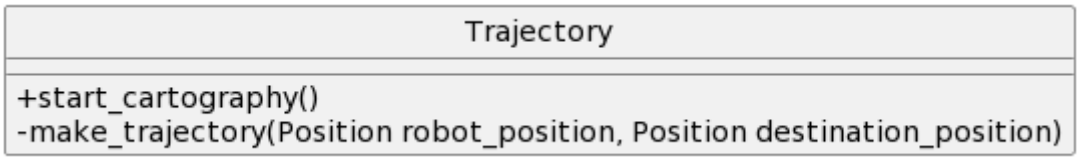


Diagramme de classe de la classe Trajectory

2.3.3.1. Philosophie de conception

La classe Trajectory calcule la trajectoire la plus optimisée entre deux points : entre la position du robot et le clique de l'utilisateur sur la carte tout en se préoccupant des obstacles sur son chemin. Elle modifie la matrice (ou grille) grâce à son algorithme.

2.3.3.2. Description structurelle

Services offerts :

- **(+)start_trajectory() : void :** Méthode publique permettant de lancer le processus de déplacement du robot en suivant une trajectoire optimisée.
- **(-)make_trajectory(Position robotPosition, Position destination_position) : void :** Méthode privée permettant, à l'aide de la position du robot et de la destination voulue, de modifier la matrice Map et d'ajouter la trajectoire optimisée à la matrice. Elle fait appel aux méthodes précédentes.

La liste des mouvements possibles :

- Forward : Le robot avance d'une case en avant.
- Left : le robot doit tourner de 90° à gauche, sur lui-même.
- Right : le robot doit tourner de 90° à droite, sur lui-même.

Algorithme de la méthode calculated_optimized_trajectory() : Utilisation de l'Algorithme A* (A star) avec l'Heuristique de Manhattan.

Choix de l'Algorithme :

- L'algorithme A* est sélectionné pour son efficacité dans la recherche de chemin optimisé.
- Il est combiné avec l'heuristique de la distance de Manhattan, adaptée pour des mouvements uniquement horizontaux et verticaux.

Contexte d'Utilisation :

- L'espace est modélisé sous forme de matrice, semblable à une grille, où chaque cellule représente une position potentiellement accessible par le robot.

Définition de l'Heuristique :

- L'heuristique de Manhattan estime le coût restant pour atteindre la destination à partir d'un point donné.

Sa formule est :

$d(p,q)=|px-qx|+|py-qy|$,

où p et q sont des points dans la grille avec leurs coordonnées respectives px,py et qx,qy.

Fonctionnement de l'Algorithme A* :

a. Initialisation :

L'algorithme commence par placer le point de départ dans une liste ouverte de nœuds à explorer.

b. Choix du Nœud :

À chaque itération, l'algorithme sélectionne le nœud ayant le coût total le plus faible (coût déjà engagé + heuristique) dans la liste ouverte.

c. Exploration :

- Il explore le nœud choisi en examinant ses voisins directs (gauche, droite, haut, bas)
- Si un voisin est le point d'arrivée, l'algorithme s'achève.

Sinon, il ajoute les voisins non explorés dans la liste ouverte.

d. Terminaison :

L'algorithme se termine quand le point d'arrivée est atteint ou si la liste ouverte est vide (pas de chemin possible).

2.3.4. [IHM] La classe Map

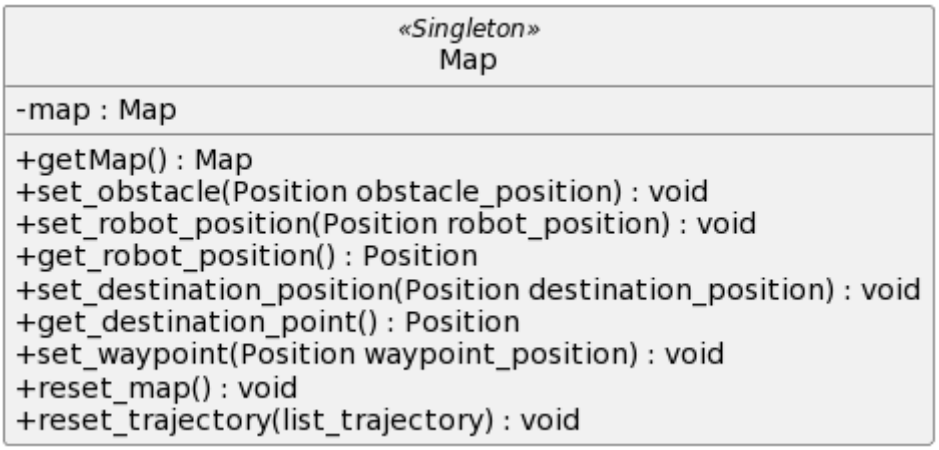


Diagramme de classe de la classe Map

2.3.4.1. Philosophie de conception

La classe Map est un Singleton, une seule instance d'une matrice permettant d'enregistrer la position du robot, la destination de l'utilisateur, les obstacles et la trajectoire entre le robot et la destination.

2.3.4.2. Description structurelle

Attributs

- **map : Map** : matrice

Services offerts

- **(+)getMap()** : Map : Méthode publique pour obtenir la map à l'aide de son instance
- **(+)set_obstacle(Position obstacle_position) : void** : Méthode publique permettant de définir une case de la grille (ou matrice) comme un obstacle.
- **(+)set_robot_position(Position robot_position) : void** : Méthode publique permettant de définir une case de la grille (ou matrice) la position du robot.
- **(+)get_robot_position() : void** : Méthode publique permettant d'obtenir la position du robot.
- **(+)set_destination_position(Position destination_position) : void** : Méthode publique permettant de définir une case de la grille (ou matrice) la destination voulue par l'utilisateur.
- **(+)get_destination_point() : Position** : Méthode permettant d'obtenir la destination choisie par l'utilisateur.
- **(+)set_waypoint(Position waypoint_position) : void** : Méthode publique permettant de définir une case de la grille (ou matrice) comme un point dans la trajectoire que le robot devra prendre pour arriver à destination.
- **(+)reset_map() : void** : Méthode publique permettant de réinitialiser la matrice.
- **(+)reset_trajectory(list_trajectory) : void :** Méthode publique permettant de supprimer la trajectoire créé dans la Map

Dans la matrice, chaque case correspond à un élément spécifié par un numéro :

- PASSAGE = 1 # Zone libre où le robot peut passer.
- OBSTACLE = 0 # Zone avec un obstacle où le robot ne peut pas passer.
- UNKNOWN = 2 # Zone inconnue, considérée comme un obstacle dans ce contexte.
- WAYPOINT = 3 # Point de passage intermédiaire sur la trajectoire du robot.
- DESTINATION = 4 # Point de destination du robot.
-

Définition des différentes orientations possibles du robot :

- ROBOT_ORIENTATION_EST = 5
- ROBOT_ORIENTATION_NORD = 6
- ROBOT_ORIENTATION_SUD = 7
- ROBOT_ORIENTATION_OUEST = 8
-

Orientation initiale du robot :

- ROBOT = ROBOT_ORIENTATION_EST
-

2.3.5. [Objet] La classe Ultrasound

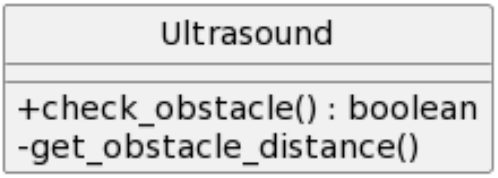


Diagramme de classe de la classe Ultrasound

2.3.5.1. Philosophie de conception

La classe Ultrasound permet d'évaluer la distance séparant le robot de l'obstacle et indique si un obstacle se trouve à moins de **12cm.**

2.3.5.2. Description structurelle

Attributs

N.A.

Services offerts

- **(+)check_obstacle() : boolean** : Méthode publique indiquant si un obstacle est présent à moins de 12cm du robot. Retourne un booléen.
- **(-)get_distance_obstacle() : distance** : Méthode privée retournant la distance en centimètres entre le robot et le prochain obstacle se situant dans la zone couverte par le capteur ultrason.

2.3.4. [Objet] La classe Motor

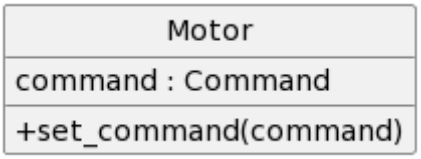


Diagramme de classe de la classe Motor

2.3.4.1. Philosophie de conception

La classe Motor s'occupe de déplacer le robot.

2.3.4.2. Description structurelle

Attributs

- **command : command :** Type de commande de déplacement envoyée par Cute à Carto.

Services offerts

- **(+)set_command(command) : void :** Méthode publique permettant de déplacer le robot vers une direction donnée.

2.3.6. [Objet] La classe Pilot

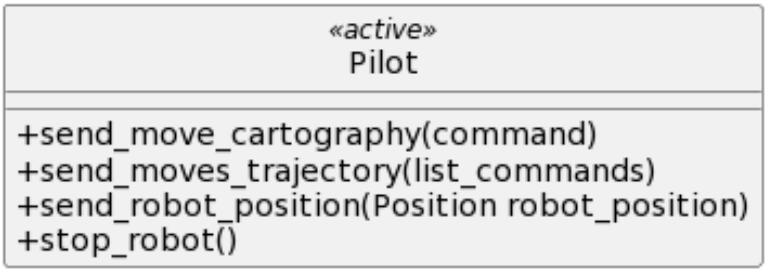


Diagramme de classe de la classe Pilot

2.3.6.1. Philosophie de conception

La classe Pilot permet de piloter le robot à la demande de GUI et de notifier des obstacles détectés par l'ultrason.

2.3.6.2. Description structurelle

Attributs

N.A

Services offerts

- **(+)send_move_cartography(command) : void :** Méthode publique récupérant l'instruction à exécuter pour pouvoir cartographier.
- **(+)send_moves_trajectory(list_commands) : void :** Méthode publique récupérant la liste d'instructions à exécuter pour pouvoir faire un déplacement autonome.
- **(+)send_robot_position(Position robot_position) : void :** Méthode publique récupérant la position du robot sur Map.
- **(+)stop_robot() : void :** Méthode publique permettant l'arrêt du robot, il ne prendra pas en compte les commandes de déplacements qui lui ont été envoyées et qui n'ont pas encore été exécutées.

2.4 Protocole de communication

2.4.1. Protocole de communication entre Carto et Cute

2.4.1.1. Formalisation du protocole de communication entre Carto et Cute

Les communications entre les deux composants logiciels suivront le même protocole de communication, défini de la forme suivante :

Size	Msg	Data

- **Size :** Taille totale du message, comprenant la taille des champs Msg et Data. Size sera envoyée sur 2 octets ce qui laisse la possibilité d'avoir une taille maximale de 65535 octets de données à envoyer.
- **Msg :** Type du message transmis, il sera envoyé sous la forme de deux octets ce qui laisse 65535 possibilités de types différents.
- **Data :** Il s'agit de la donnée à envoyer. Sa taille est variable (de 0 à 65533 octets) c'est pourquoi le champ Size est nécessaire.

Size et Msg constituent l'entête de la trame envoyée, cet header de 4 octets reste constant en taille. Les commandes sont formatées en hexadécimal. Les logs sont formatés en chaine de caractères ascii.

2.4.1.2. Types de messages

Le tableau ci-dessous détaille les types de messages définis dans le protocole de communication.

Identifiant en hexadécimal	Msg	Commentaires
0x0100 (256)	SEND_MOVES_TRAJECTORY	Cute envoie une liste de commande de direction pour la trajectoire à Carto
0x0200 (512)	SEND_MOVE_CARTOGRAPHY	Cute envoie une liste de commande de direction pour la cartographie à Carto
0x0300 (768)	MOVE_DONE	Carto inidque à Cute que le déplacement a été réalisé
0x0400 (1024)	SET_OBSTACLE_POSITION	Cute indique à Carto qu'il y a un obstacle à une certaine position
0x0500 (1280)	SET_ROBOT_POSITION	Carto indique la position du robot à Cute
0x0600 (1536)	STOP_ROBOT	Cute envoie une commande d'arrêt du robot à Carto

0x0700 (1792)	SEND_ROBOT_POSITION	Cute envoie les coordonnées du robot à Carto
0x0800 (2048)	ROBOT_POSITION_RECEIVED	Carto indique à Cute qu'il a bien reçu la position du robot

2.4.1.3. Détails des messages

Sur les 8 types de messages, 3 messages n'ont pas de données à embarqués et ne sont que de simples commandes :

- ROBOT_POSITION_RECEIVED
- MOVE_DONE
- STOP_ROBOT.

Pour les 5 autres types de messages, le tableau ci-dessous détail les données envoyées par type de message :

Identifiant	**Msg**	**Data (1 octet = 1 case)**		
0x0100 (256)	SEND_MOVES_TRAJECTORY	***size_list_command***	***command_type***	
0x0200 (512)	SEND_MOVE_CARTOGRAPHY	***command_type***		
0x0400 (1024)	SET_OBSTACLE_POSITION	***coord_x***	***coord_y***	
0x0500 (1280)	SET_ROBOT_POSITION	***coord_x***	***coord_y***	
0x0700 (1792)	SEND_ROBOT_POSITION	***coord_x***	***coord_y***	***direction***

2.4.1.4. Exemples de messages

2.4.1.4.1. SEND_MOVES_TRAJECTORY

Cute envoie des commandes de déplacement à Carto, à travers la classe Trajectory. Les trames suivantes sont transmises :

Size	**Msg**	**Data**	
0x00 0x04	0x01 0x00	0x03	0x00
0x00 0x04	0x01 0x00	0x03	0x01
0x00 0x04	0x01 0x00	0x03	0x02

Ici, SEND_MOVES_TRAJECTORY est une liste de trame de 6 octets :

- **Size** vaut 0x04 (4) ce qui correspond aux deux octets de Msg plus l'octet de Data.
- **Msg** vaut 0x0100 ce qui correspond à la requête SEND_MOVES_TRAJECTORY.
- **Data** :
  (1er octet) vaut 0x03 (3), ce qui correspond au nombre de trame qui seront envoyées pour SEND_MOVES_TRAJECTORY.
  (2ème octet) première trame : 0x00 (0) → vers l'avant. (FORWARD)
  deuxième trame : 0x01 (1) →rotation de 90° vers la gauche (RIGHT)
  troisième trame : 0x02 (2) →rotation de 90° vers la droite (LEFT)

2.4.1.4.2. SEND_MOVE_CARTOGRAPHY

Cute envoie une commande de déplacement à Carto, à travers la classe Cartography. La trame suivante est transmise :

Size	Msg	Data
0x00 0x03	0x02 0x00	0x00

Ici, SEND_MOVE_CARTOGRAPHY est une trame de 5 octets :

- **Size** vaut 0x03 (3) ce qui correspond aux deux octets de Msg plus l'octet de Data.
- **Msg** vaut 0x0100 ce qui correspond à la requête SEND_MOVE_CARTOGRAPHY.
- **Data** 0x00 (0) -> vers l'avant. (FORWARD)

2.4.1.4.3. MOVE_DONE

Après avoir effectué une commande de déplacement envoyé par Cartography, Carto envoie à Cute une trame pour confirmer que la commande a bien été réalisée :

Size	Msg	Data
0x00 0x02	0x03 0x00	

Ici, MOVE_DONE est une trame de 4 octets :

- **Size** vaut 0x02 (2) ce qui correspond aux deux octets de Msg plus l'octet de Data.
- **Msg** vaut 0x0300 ce qui correspond à la requête MOVE_DONE.

2.4.1.4.4. SET_OBSTACLE_POSITION

Avant d'avoir effectué une commande de déplacement FORWARD envoyé par Cartography, Carto envoie à Cute une trame si Ultrasound indique la présence d'un obstacle :

Size	Msg	Data
0x00 0x04	0x04 0x00	0x09 0x15

Ici, SET_OBSTACLE_POSITION est une trame de 6 octets :

- **Size** vaut 0x04 (4) ce qui correspond aux deux octets de Msg plus deux octets de Data.
- **Msg** vaut 0x0400 ce qui correspond à la requête SET_OBSTACLE_POSITION.
- **Data** vaut : 0x09 (9) → coordonnées en x = 9 ;
  0x15 (21) →coordonnées en y = 21.

2.4.1.4.5. SET_ROBOT_POSITION

Après avoir effectué une commande de déplacement FORWARD envoyé par Cartography, Carto envoie à Cute une trame pour indiquer la nouvelle position du robot, si celle-ci a changé :

Size	Msg	Data
0x00 0x04	0x05 0x00	0x21 0x02

Ici, SET_ROBOT_POSITION est une trame de 6 octets :

- **Size** vaut 0x04 (4) ce qui correspond aux deux octets de Msg plus deux octets de Data.
- **Msg** vaut 0x0500 ce qui correspond à la requête SET_ROBOT_POSITION.
- **Data** vaut : 0x21 (33) → coordonnées en x = 33 ;
  0x02 (2) →coordonnées en y = 2.

2.4.1.4.6. STOP_ROBOT

Cute indique à Carto que le robot doit se stopper s'il effectue un mouvement au moment de la réception du message, ou si des commandes de déplacement sont prêtes à être réalisées. La trame est la suivante :

Size	Msg	Data
0x00 0x02	0x06 0x00	

Ici, STOP_ROBOT est une trame de 4 octets :

- **Size** vaut 0x02 (2) ce qui correspond aux deux octets de Msg plus l'octet de Data.
- **Msg** vaut 0x0600 ce qui correspond à la requête STOP_ROBOT.

2.4.1.4.7. SEND_ROBOT_POSITION

Avant de commencer à envoyer des commandes de déplacements au robot pour la cartographie, Cute envoie la position du robot dans la matrice à Carto. La trame est de la forme suivante :

Size	Msg	Data
0x00 0x05	0x07 0x00	0x017 0x14 0x00

Ici, SEND_ROBOT_POSITION est une trame de 6 octets :

- **Size** vaut 0x05 (5) ce qui correspond aux deux octets de Msg plus deux octets de Data.
- **Msg** vaut 0x0700 ce qui correspond à la requête SEND_ROBOT_POSITION.
- **Data** vaut : 0x17 (23) → coordonnées en x = 23 ;
  0x14 (20) →coordonnées en y = 20.
  0x00 (0) →direction = +y (axe des ordonnées)

2.4.1.4.8. ROBOT_POSITION_RECEIVED

Carto indique à Cute que la trame indiquant la position du robot a bien été reçue, c'est donc une réponse à SEND_ROBOT_POSITION. La trame est la suivante :

Size	Msg	Data
0x00 0x02	0x08 0x00	

Ici, ROBOT_POSITION_RECEIVED est une trame de 4 octets :

- **Size** vaut 0x02 (2) ce qui correspond aux deux octets de Msg plus l'octet de Data.
- **Msg** vaut 0x0800 ce qui correspond à la requête ROBOT_POSITION_RECEIVED.