Pour l’évitement :

## Passer du scan du lidar à une liste de point :

from\_scan\_to\_points

* en entrée: la liste des valeurs renvoyé par la valeur de lidar.scan.distances
* en sortie : une dictionnaire associant à chaque angle la distance des points le plus proches

forme du dict : key : angle en rad / value : distances lidar points en cm

* Remarque : pourquoi pas ajouter des paramètre facultatif permettant d’avoir les angles en degré ou radian (mis à radian par défaut), et un paramètre permettant de gérer l’unité des distances (mis en cm par défaut)

## Calculateur de coordonnées

coords\_relative\_calculator

* Entrée : angle point lidar / distances
* Sortie : coordonnées relatives de points (coordonnées du point par rapport au lidar)

## Calculer la position relative des points

points\_relative\_coords\_calulator

* En entrée : dictionnaire angle <-> distance
* En sortie : liste de points

Le but de cette fonction est de calculer la position (x, y) par rapport au robot de chaque point mesuré par le lidar.

En retour, il faut une liste de tuple, chaque tuple correspondra aux coordonnées d’un point sous la forme (x, y). Il faut utiliser coords\_relative\_calculator.

## Calculer la position absolue des points

absolute\_coords\_calulator

* Entrée : coordonnées d’un point tuple x, y -> (x, y) / position du robot (x, y, theta)
* Sortie : coordonnées absolues du point (x, y)

Le but est de transposer les coordonnées du point en position absolue càd dire de trouver l’emplacement de ce point sur la table à partir de sa position relative.

## Calculer la position des points dans l’espace

points\_absolute\_coords\_calulator

* Entrée : liste des coordonnées relatives des points / position du robot (x, y, theta)
* Sortie : liste des coordonnées absolue des points

Il faut utiliser absolute\_coords\_calulator.

## Former des clusters de points :

clustering\_detection

* Entrée : taille cluster / distance cluster
* Sortie : une liste de liste contenant les coordonnées des points extérieur aux clusters.

Le but de cette fonction est de faire du clustering de points. L’idée est d’associer les points en groupes de points pour identifier la présence d’objet (par exemple un robot).

Cette fonction permettra d’éviter tous les faux positifs, comme ça, s’il y a un point unique très proche du lidar, il ne sera pas compté comme étant un danger car il peut s’agir d’une simple erreur de lecture.

Le paramètre taille\_cluster sera là pour définir un nombre de points minimum permettant de former un cluster par exemple 10 -> dès qu’un regroupement de 10 (ou plus) points proches sera détecté, un cluster sera alors crée.

Le paramètre distance cluster est là pour donner une limite à partir de laquelle 2 points seront considéré comme proche. Utile pour savoir si les points sont suffisamment proches pour créer un cluster.

Pour la sortie : une fois un cluster crée, il faut identifier tous les points extérieurs de ce cluster et les mettre dans une sous liste. Le but est qu’en sortie on puisse récupérer le contour de tous les clusters pour pouvoir les afficher sur le GUI de Noah par exemple.

* Remarque : comme précédemment tu peux ajouter des paramètres optionnels permettant de gérer les unités des distances et des angles, le but est de faire les fonctions les plus génériques possibles.

## Arrêt urgences par proximité :

proximity\_urgency\_stop

* En entrée, distance critique d’arrêt d’urgence
* En sortie, true / false

Le but de cette fonction est d’envoyer au robot l’ordre de s’arrêter lorsqu’un cluster d’une distance inférieure au paramètre donnée en entrée est détecté.

Tu pourras donc appeler la fonction clustering détection pour récupérer tous les clusters proches.

Lorsqu’un cluster est détecté il faut lancer un nouveau thread qui va envoyer en boucle la « stop » sur le topic d’arrêt du robot. Pour lancer le nouveau thread il faut lui donner une fonction target, pour ça tu peux créer une sous fonction « \_send\_stop » qui va s’occuper de ça. Il faut évidement que cette fonction exit le thread lorsque le message a bien été reçu par la teensy. Le pb c’est qu’on est en UDP donc on a pas d’accusé de réception, pour ça j’ai créé dans Intercom\_tools des fonctions qui s’occupent de ça (il faut par contre déclarer un topic dans lequelle le robot va envoyer en boucle si il est en arrêt ou non )

La fonction doit renvoyer true ou false en fonction de la détection ou non d’un cluster (ça peut tjrs être utile)