Lidar

RAPPORT TECHNIQUE V1 - CDR 2021

Elio VICENTE | CDR-2021 | Août 2021

Introduction

Ce rapport décrit l'usage que nous faisons du Lidar : RPLidar-A1. Ce composant mis au point par la marque Slamtec constitue l'option la moins chère du marché pour se doter d'un Lidar. Le fonctionnement d'un lidar est relativement simple, un faisceau laser est émis depuis le dispositif, en mesurant le temps que met le faisceau à revenir, la distance avec l'objet éventuel est déduite. Le tout est monté sur une plateforme rotative permettant au lidar de scanner un plan bidimensionnel. Pour avoir une idée plus précise des données produites, voici une vidéo qui pourrait vous intéresser :

https://www.youtube.com/watch?v=lstwx-55hgk

Lors de l'édition 2021 de la Coupe de France de Robotique, ce lidar a été utilisé afin d'éviter d'entrer en collision avec d'autres robots. En effet, lors de cette édition de la coupe, l'ensemble des robots présents sur le terrain disposaient d'une balise située à hauteur du lidar, simplifiant ainsi la détection. En cas de détection d'un robot à proximité immédiate, le robot était ainsi capable de prendre la décision de s'arrêter.

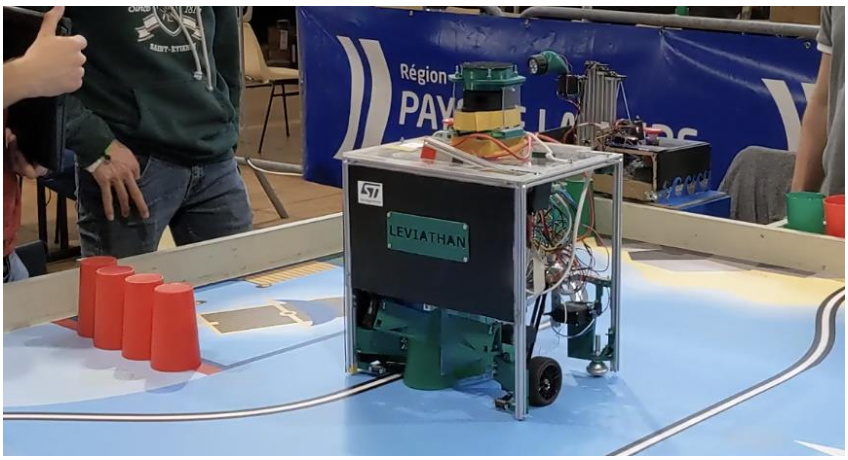

Figure 1: Robot Léviathan sur une table d'essai à la Roche Sur Yon

Développement

FONCTIONNEMENT DU COMPOSANT

Les éléments essentiels pour le fonctionnement de ce composant sont les suivants :

- Une alimentation
- Une connexion série, afin de récupérer les données produites.

Les spécifications sur l'alimentation du composant sont assez précises et il convient de ne pas se tromper sur l'alimentation du scanner system… au risque de griller le lidar.

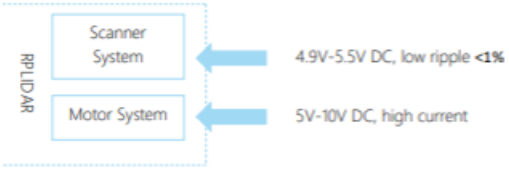

Figure 2: Spécifications d'alimentation

Lors de notre édition nous n'avons pas eu à réfléchir sur l'alimentation du lidar car nous avons utilisé le module USB de Slamtec qui permet d'alimenter et de récupérer les données lidar en utilisant un seul et même câble USB.

Figure 3: Module USB Slamtec

Mécaniquement parlant l'installation du lidar à la bonne hauteur et à la bonne place a été réalisée à l'aide :

- De profilés en aluminium de la marque MakerBeam.
- De pièces imprimées en 3D (jonction lidar/markerbeam)

Sachant qu'un tag est placé au-dessus du lidar lors des matchs, une surface en velcro située à une hauteur précise est requise. Pour cela un assemblage entre des pièces imprimées en 3D et du plexiglass transparent a été réalisé.

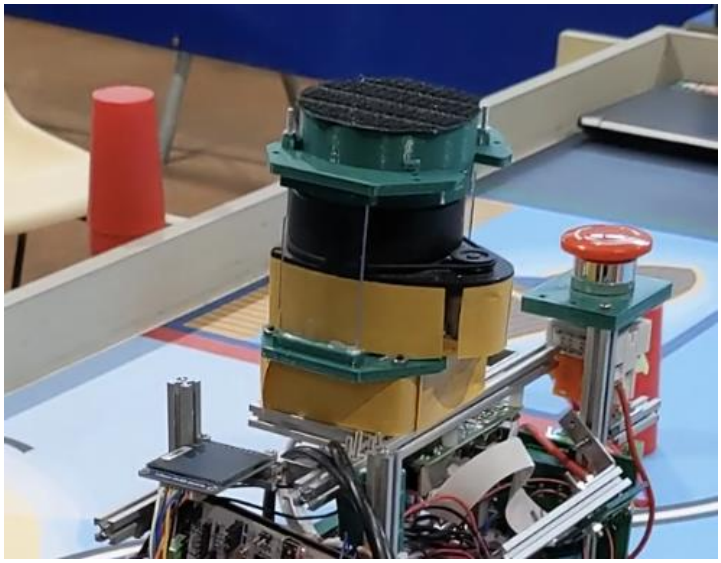

Figure 4: Lidar assemblé dans sa configuration pré-match

Après avoir discuté des aspects physiques, essayons d'avoir une vue plus globale du système en y incorporant la partie logique logicielle.

Figure 5: Branchement Lidar/Raspberry

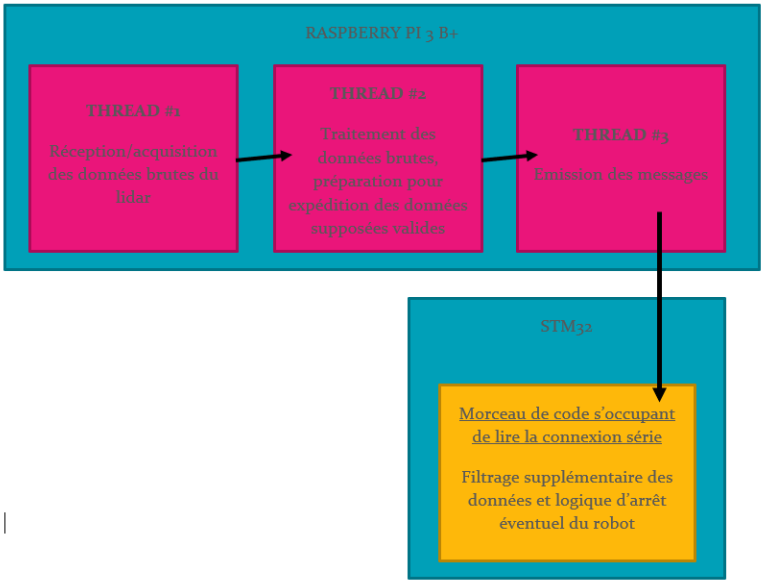

Figure 6: Logique grossière menant à l'arrêt éventuel du robot, dans un but pédagogique de nombreux éléments ne sont pas mentionnés dans le schéma. Voir les parties suivantes pour plus de détails.

CHOIX DU COMPOSANT

Nous disposions grâce aux mandats précédents et à M. Marques (merci à lui 😊) de 2 exemplaires de RP-Lidar A1, nous n'avons pas jugé opportun de réaliser une upgrade de ce composant lors de cette édition.

INTÉGRATION DU COMPOSANT

Comme mentionné dans les parties précédentes un seul câble relie ce composant au reste du robot, un câble USB qui se branchait sur port USB d'une Raspberry PI 3B.

Le traitement des données est effectué par un script python, celui-ci est disponible sur le Gitlab d'AREM :

https://gitlab.emse.fr/arem/coupe-de-france-2021/lidar

- **Exécution du logiciel**

Modes de fonctionnement principaux	**Fonction**
Lancement du script au démarrage de la raspy.	Permet de ne pas avoir besoin de son PC ni de se connecter à la raspberry pour mettre en place le système. Pratique avant un match.
Lancement du script manuellement depuis un terminal.	En cas de crash quelconque nous pouvons visualiser les logs et relancer le script.

- **Description du logiciel**

Nom du fichier principal	master.py
Rôle du fichier principal	Initialiser les liaisons, lancer les différents threads nécessaires qui sont généralement présents dans le dossier module/

- **Modules principaux appelés par le logiciel et leurs rôles**

Rôle	**Module**	**Description**
Procéder à l'acquisition des données lidar	module/lidar.py	Un thread est lancé pour effectuer l'acquisition, ce thread exécute la fonction cooker()
Procéder au traitement des données lidar	module/lidar.py	Un thread est lancé pour effectuer le traitement, ce thread exécute la fonction consumer()
Réceptionner les messages en provenance de la STM32	module/threads.py	Un thread est lancé pour effectuer la réception des messages, ListenerThread()
Traiter les messages dans la boite d'envoi à envoyer à la STM32	module/threads.py	Un thread est lancé pour effectuer la réception des messages, CommandThread()

Traiter les données caméra *(Désactivé pendant la coupe)*	module/gcolor.py	Un thread est lancé pour effectuer l'acquisition des couleurs, gcolor()

- **Procéder à l'acquisition des données Lidar :**

```
def cooker(name, exit_event):
        lidar = RPLidar('/dev/ttyUSB0')
        lidar.reset()
        time.sleep(1)
        lidar.clean_input()
        info = lidar.get_info()
        print(info)
        health = lidar.get_health()
        print(health)
        try:
                for i, scan in enumerate(lidar.iter_scans()):
                        if exit_event.is_set():
                                break
                        q.put(scan)
        finally:
                lidar.stop()
                lidar.stop_motor()
                lidar.disconnect()
```

Voici le morceau de code qui effectue l'acquisition des données lidar. Les lignes jaunes représentent du code qu'il convient d'insérer dans cet ordre précis afin d'éviter des bugs étranges (le genre de bug qui fait passer des nuits blanches la veille des matchs). A noter que ces instructions et plus particulièrement l'ordre dans lequel je les place est introuvable dans documentation officielle.

Revenons à nos moutons. Le code qui est important ici est la première ligne et le contenu de la boucle for. La première ligne est l'ouverture de la liaison avec le Lidar, une fois cette dernière ouverte on peut parcourir les données au fur et à mesure qu'elles sont générées par le lidar. A chaque fois qu'un scan complet (360°) est réalisé le scan est mis dans une file d'attente (q), cette dernière est traitée au fur et à mesure par le thread de traitement des données lidar.

Le traitement n'est pas effectué directement par le thread d'acquisition pour ne pas prendre du retard sur le « fetch » des données présentes dans le buffer du lidar. En effet si l'on prend du retard sur le « fetch » non seulement nos données seront moins à jour mais on risque de provoquer un overflow du buffer du lidar provoquant un crash.

- **Procéder au traitement des données Lidar :**

```
scanIndex = 0
objc = []
while True:
        time.sleep(0.005)
        scanIndex += 1
        scan = q.get()
        for mes in scan:
                (r, a) = (mes[2], mes[1])
                if(r > maxradius):
                        continue
                if(r < minradius):
                        continue
                a = 180 -a
                #print('sharedx %d'%module.dshare.myPos[0])
                #print('sharedy %d'%module.dshare.myPos[1])
                xlidarref = r*np.cos(a*np.pi/180)
                ylidarref = r*np.sin(a*np.pi/180)
                angleabsolu = module.dshare.myPos[2]/1000
                x = module.dshare.myPos[0] + xlidarref*np.cos(angleabsolu) - ylidarref*np.sin(angleabsolu)
                y = module.dshare.myPos[1] + ylidarref*np.cos(angleabsolu) + xlidarref*np.sin(angleabsolu)
                xmin = exclusionZoneFromBorder
                xmax = xterrainmax - exclusionZoneFromBorder
                ymin = exclusionZoneFromBorder
                ymax = yterrainmax - exclusionZoneFromBorder
                if(x < xmin or x > xmax or y < ymin or y > ymax):
                        continue
                #print('x: %d'%x)
                #print('y: %d'%y)
                fflag = False
                for element in objc:
                        if(isNearPoint(x, y, element.x, element.y)):
                                fflag = True
                                mid = getMidPoint(x, y, element.x, element.y)
                                element.x = mid[0]
                                element.y = mid[1]
                                element.count += 1
                                break
                if(not fflag):
                        objc.append(ObjTrack(x, y))
        if(scanIndex % nbrScanToRegroup ):
                module.dshare.obj = objc
                for element in module.dshare.obj:
                        #print('ex: %d'%element.x)
                        #print('ey: %d'%element.y)
                        if(element.count > minPoints):
                                queue.put((Order.MSG_OBJ_POS, [element.x, element.y]))
                objc = []
        #print('%s ate'%(name))
        #print(q.qsize())
```

Voici un extrait du code de traitement des données lidar.

La procédure est assez simple :

1. On prend un scan dans la Queue
2. On traite chaque distance mesurée dans ce scan
3. Pour chaque mesure, on applique un filtrage en distance
4. Pour chaque mesure, on applique les changements de référentiel nécessaires
5. Certains changements de référentiels nécessitent la connaissance de la position actuelle du robot relativement à la table de jeu.
6. Des filtres additionnels permettent un filtrage des points qui sont en dehors de l'aire de jeu.

Le code qui suit ces opérations consiste essentiellement à regrouper les points entre eux afin d'augmenter la précision des mesures et de limiter l'impact des données parasites.

Au bout d'un moment si un point est la conséquence du regroupement de nombreux points à son voisinage, nous décidons de mettre dans la boite d'envoi un message de Type : (MSG_OBJ_POS, [coordx, coordy]). Ce message sera alors envoyé à la STM32 lorsque le thread s'occupant d'expédier les messages de la boite d'envoi sera libre.

En résumé :

Numéro	**Étape**	**Description**
1	Filtrage du point en distance par rapport au lidar	Les points situés à plus de maxradius et les points situés à moins de minradius sont éliminés.
2	Filtrage du point s'il est en dehors du terrain*	Les points situés en dehors du terrain sont exclus, et les points situés dans une zone tampon au niveau du bord du terrain sont exclus.
3	Regroupement des points proches	Si le point vu se situe à proximité d'un point déjà aperçu lors des X scans précédents ou du scan en cours alors il est fusionné avec ce dernier. De plus, lors de la fusion, le point en mémoire est translaté au barycentre des deux points. Un compteur permet de savoir combien de points ont été fusionnés pour former le point actuellement en mémoire.
4	Filtrage pour expédition	Une fois les X scans réalisés, les points en mémoires ayant résulté de la fusion de plus de n points sont mis en forme pour être envoyer.
5 (**extérieur au système**)	Filtrage à l'arrivé des données sur la STM32	Il s'agit d'une étape optionnelle, mais qui permet de filtrer des éléments annexes non prévus par notre système. Notre système est simplement là pour fournir des données propres et cohérentes et n'a pas vocation à contenir de la logique propre au match. Ce filtrage relatif à la logique du match a été utilisé pour : - Ne pas considérer les robots qui seraient de notre côté du terrain - Ne pas considérer la girouette.

Conclusion

Le code est fortement multi-threadé et peut paraitre un peu rebutant, j'espère que ce rapport pourra aider l'éventuel repreneur du code. Il faudra dans la prochaine édition permettre un lancement du script via un bouton physique à mon avis et avoir un mini-écran connecté à la Raspberry pour faciliter le debug et éviter les incompréhensions. L'écran pourra également être utilisé pour s'assurer que le script tourne bien comme il faut avant un match.

Il peut être envisagé d'utiliser le lidar à d'autre fins dans les prochaines éditions :

- Détection des éléments de jeu par exemple en plaçant le lidar au niveau du sol
- Détection du mat central afin de se localiser ? (il faudra pour le coup upgrade le lidar à mon avis)
- Il est également relativement aisé d'estimer la trajectoire des robots adverses, cette information pourrait être utile pour la mise en place de stratégies dynamiques.