Robots LEGO EV3 : localisation par caméras Optitrack

1 Présentation

1.1 Objectif

Dans ce travail, vous allez acquérir des mesures de position et d'orientation à l'aide des caméras Optitrack. Pour cela, vous utiliserez un package Python qui fera l'interface entre les caméras et votre programme. L'objectif est de prendre en main cet interface et de réaliser un programme capable d'enregistrer la trajectoire de votre robot.

1.2 Interface Python pour les caméras Optitrack

1.2.1 Description

Le nouveau système OptiTrack https://optitrack.com vient en remplacement des anciennes caméra dont la précision été assez faible.

Les caméras https://optitrack.com/cameras/primex-22/ fontionnent via Ethernet (RJ45) en PoE. Elles sont réparties tout autour de l'espace de travail des robots. A l'aide de LED infrarouge, elles localisent dans l'espace des sphères de réflexion.

Trois legos sont équipées de sphères définissant une empreinte unique. **Veuillez ne pas y toucher !** Le logiciel Motive exécuté sur le PC ENSEM est capable de reconnaitre ces empreintes et ainsi d'identifié chaque robot. D'autre part, il mesure la localisation du centre de l'essieu et l'orientation du robot dans l'espace. Ces informations sont donnés dans le « repère Motive » représenté en bleu sur la Figure 1.

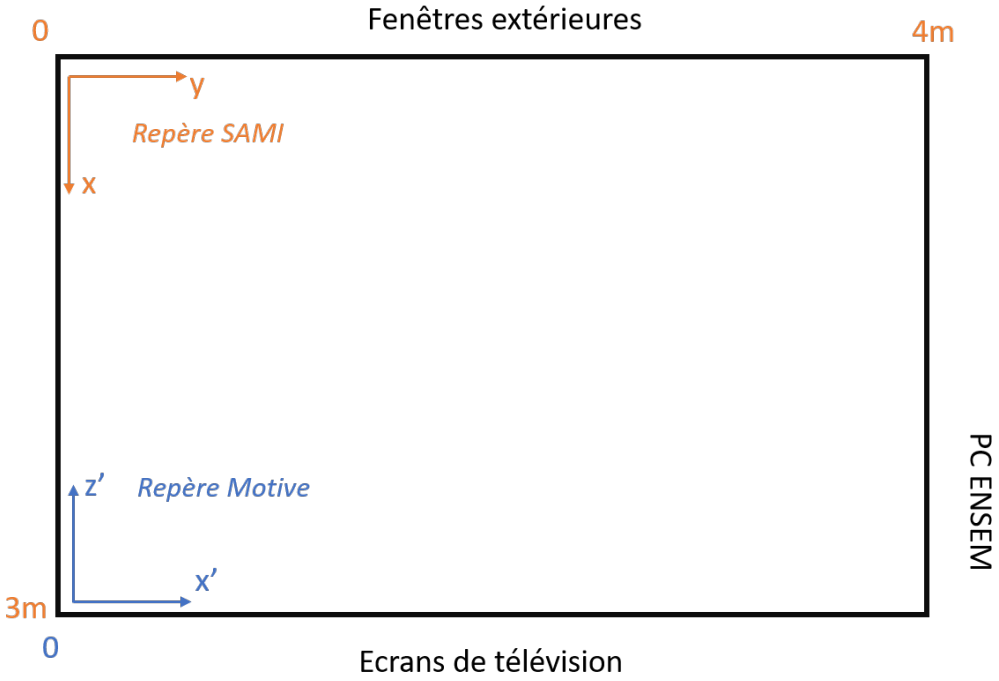

Figure 1 – Schéma de la salle SAMI avec les deux repères : SAMI (orange) et Motive (bleu)

Les caméras Optitrack communiquent sur le réseau en s'appuyant sur un protocole propriétaire nommé Natnet. Afin de récupérer et décoder ces trames, il est nécessaire de : i) déployer un client UDP sur votre

machine, ii) décrire les méthodes de décodage pour les messages natnet, iii) traiter les données brutes pour se ramener dans votre contexte applicatif.

Dans le cadre de ce TP, ces programmes vous sont fournis via l'archive `Optitrack.zip`. Cette interface s'appuie sur trois objets : le client Natnet (`NatnetClient()`), les messages Natnet (`Packet()`) et le noeud d'interface (`MocapNode()`).

Afin de faciliter votre travail, les coordonnées sont convertis dans le « repère SAMI » représenté en orange sur la Figure 1.

1.2.2 Installation

Pour installer l'interface, veuillez décompresser le contenu de l'archive `Optitrack.zip` à la racine de votre programme. En supposant votre programme nommé `myprogram.py`, votre répertoire doit ressembler à :

— ./src/
— ./Tests/
— `config.ini`
— `myprogram.py`

Afin de tester l'installation, veuillez exécuter le programme testAll.py depuis la racine. Par exemple, ouvrez votre répertoire dans VisualStudio Code. Naviguez jusqu'à ./Tests/testAll.py. Puis, pressez F5. L'ensemble des tests doit être validés que vous soyez connectés au réseau ou non.

1.2.3 Utilisation

Pour utiliser l'interface, il est nécessaire de renseigner votre ip et celle du serveur Optitrack dans le fichier `config.ini`. Le serveur Optitrack de la salle SAMI répond à l'addresse : « 100.64.212.150 ». Pour connaitre votre adresse ip, veuillez exécuter sous votre invite de commande : `ipconfig`. Une fois ces informations obtenues, ouvrez le fichier de configuration, ajoutez les lignes suivantes à fin du fichier (sans effacer ce qui existe déjà) en remplacant les champs par vos valeurs :

```
["MonNomDeConfig"]
srvAddr = "IPduServeur"
cltAddr = "MonIP"
```

Par exemple, pour mon PC nommé « PC6 » et travaillant sous l'ip : 100.64.212.156, j'inscris :

```
[PC6]
srvAddr = 100.64.212.150
cltAddr = 100.64.212.156
```

Dans votre programme, commencez par indiquer le chemin de l'interface :

```
import sys, os
sys.path.append(os.path.join(os.path.dirname(sys.path[0]),'src'))
```

Ensuite, importez le module `mocap_node.py` contenant la classe `MocapNode()`. Ici, nous renommerons ce module « `mcn` » :

```
        import mocap_node as mcn
```

Créez un noeud Motion Capture (Mocap) à partir de votre configuration comme un objet `MocapNode()`, soit :

```
        mymcn = mcn.MocapNode("PC6")
```

Remarquez que le constructeur prend pour argument un string correspondant au champs "MonNomDeConfig" que vous venez de renseigner.

Lancez votre noeud à l'aide de la méthode `run()` :

```
        mymcn.run()
```

A partir de cet instant, un client UDP est exécuté en tâche de fond. La communication avec les caméra est établie mais les messages ne peuvent pas encore être décryptés. Pour cela, il est nécessaire que le noeud récupère des méta-données transmises par le serveur. Ceci se réalise via la méthode `updateModelInfo()` :

```
mymcn.updateModelInfo()
```

Votre programme est maintenant prêt à acquérir des données de positions à l'aide de la méthode : `getPos2DAndYaw(name (string))`. Trois legos sont identifiés dans le noeud. Ils sont nommées : « Lego1 », « Lego2 » ou « Lego3 ». Pour obtenir la localisation du « Lego2 », on exécute :

```
pos2D, yaw = mymcn.getPos2DAndYaw("Lego2")
```

Cette méthode retourne :

— `pos2D`, un tuplet de float composé des coordonnées en mètre selon les axes x et y
— `yaw`, un float désignant l'orientation en radian selon l'axe z.

IMPORTANT : pensez à arrêter le noeud à la fin de votre programme via la méthode `stop()`. Autrement, vous risquez de ne plus pouvoir relancer votre programme une seconde fois.

```
mymcn.stop()
```

2 Travail à réaliser

2.1 Objectif

Soit un robot qui effectue une trajectoire pendant une durée T_f. On souhaite acquérir cette trajectoire avec une période d'échantillonnage $T_e = 0.1$s, puis la sauvegarder dans un fichier CSV afin de pouvoir l'afficher a posteriori (Matlab, Excel ou Numpy).

2.2 Guide

Votre travail peut s'organiser selon les points suivants :

1. Développer une boucle réalisant une action périodique toutes les T_e secondes et se terminant après T_f secondes
2. Récupérez les données de localisation à l'aide de l'interface pour les caméras
3. Sauvegardez ces données dans un fichier CSV
4. Ouvrir le fichier CSV et afficher les résultats.

Vous êtes libres d'utiliser les librairies de votre choix, comme par exemple `numpy`, `time`, `thread` ou `csv`. De même, vous avez le choix de structurer votre programme en scrypt ou classe.