Projet d’Innovation et Recherche :

FedalBot

Une image contenant personne, sport athlétique, sport, tennis

Description générée automatiquement

Une image contenant outil, automate

Description générée automatiquement

Table des matières

[1) Présentation du projet 3](#_Toc125107497)

[2) Mise en place du terrain 4](#_Toc125107498)

[3) Architecture réseau 6](#_Toc125107499)

[Idée de départ : cahier des charges et serveur Flask 6](#_Toc125107500)

[Adaptation en cours de projet 6](#_Toc125107501)

[4) Déroulement d’un match 7](#_Toc125107502)

[5) Commande des robots 8](#_Toc125107503)

[Mise en place des services pour chaque robot 8](#_Toc125107504)

[6) Arbitrage et reconnaissance d’image 9](#_Toc125107505)

[a) Paramétrage de la Jetson 9](#_Toc125107506)

[b) Prise de vue 9](#_Toc125107507)

[c) Traitement de l’image 9](#_Toc125107508)

[d) Tracking de la balle 11](#_Toc125107509)

[7) Différents axes à suivre pour poursuivre 12](#_Toc125107510)

[8) Sources 12](#_Toc125107511)

[9) Annexes 13](#_Toc125107512)

[Annexe 1 : Procédure pour mise en réseau des deux robots sur un réseau local 13](#_Toc125107513)

## Présentation du projet

Alors que les plus grands joueurs de tennis de l’histoire partent les uns après les autres il est nécessaire de préparer la relève. (Presque) toutes les fonctions remplies par un joueur de tennis peuvent l’être par un robot. Le but de ce projet sera de faire s’affronter deux robots Niryo avec des raquettes de ping-pong sur un petit terrain de tennis. On pourra modifier les règles du tennis pour simplifier le problème. Par exemple, on pourra imaginer commencer à jouer en 2D en poussant la balle avec la raquette avant de passer à la troisième dimension. Le juge de ligne (surnommé Hawkeye et constitué d’une caméra et d’un haut-parleur) annoncera les fautes lorsque la balle sortira du terrain. Les joueurs s’arrêtent lorsqu’une faute est annoncée.

Objectifs :

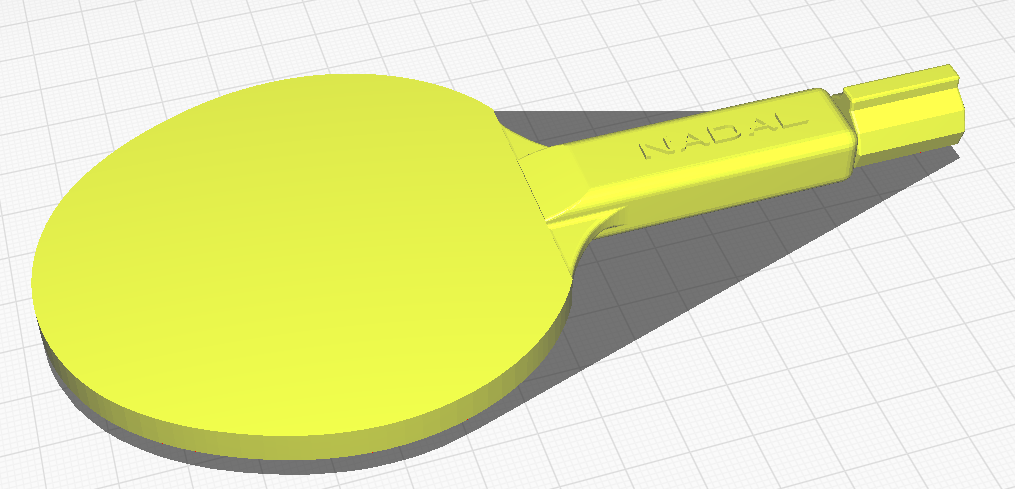
* Déployer de multiples modèles de traitements d’images, de son qui seront en interaction pour modéliser un match de tennis
* Réaliser la communication entre les robots à l’aide de serveurs Flask
* Programmer les robots afin de taper la balle
* Utiliser les modèles de machine learning (tracking, reconnaissance vocale etc.…) sur une/des cartes Jetson Nano (ou sur des ordinateurs distants et les résultats sont envoyés aux Jetson Nano qui seront les cerveaux des joueurs/arbitres)
* Développement d’un moyen de préhension pour robot Niryo permettant de manipuler une raquette de ping-pong avec le plus de degrés de libertés possibles

## Mise en place du terrain

Nous avons tout d’abord commencé notre projet en créant notre environnement de travail. Pour cela nous avons installé deux robots Niryo Ned2 et fait le marquage des terrains.

Une question s’est alors posée quant aux outils des robots. Nous étions unanimes sur le fait qu’un outil en forme de raquette devait être imprimé en 3D. Au départ nous comptions utiliser une pince fournie avec les robots.

Mais dans ce cas-là le risque que la raquette bouge lors des déplacements du robot était important. C’est pourquoi, après avoir pris les cotes nécessaires, nous avons modifié notre modèle 3D afin que la raquette soit directement couplée au robot en montage serré.



Les impressions 3D pouvaient être lancées !

Ainsi nous avons finalisé la mise en place de notre environnement de travail.

Une image contenant intérieur, plancher

Description générée automatiquement

Une image contenant texte, tableau blanc

Description générée automatiquement

## Architecture réseau

### Idée de départ : cahier des charges et serveur Flask

L’idée de départ était de suivre le cahier des charges et donc de mettre en place un serveur Flask afin de procéder aux différents échanges de données entre l’arbitre et les joueurs. De plus, ce serveur Flask nous permettait d’avoir un retour vidéo des échanges entre les robots, ainsi que l’avancée du match et le tableau des scores. Cela a donc été mis en place dès le départ.

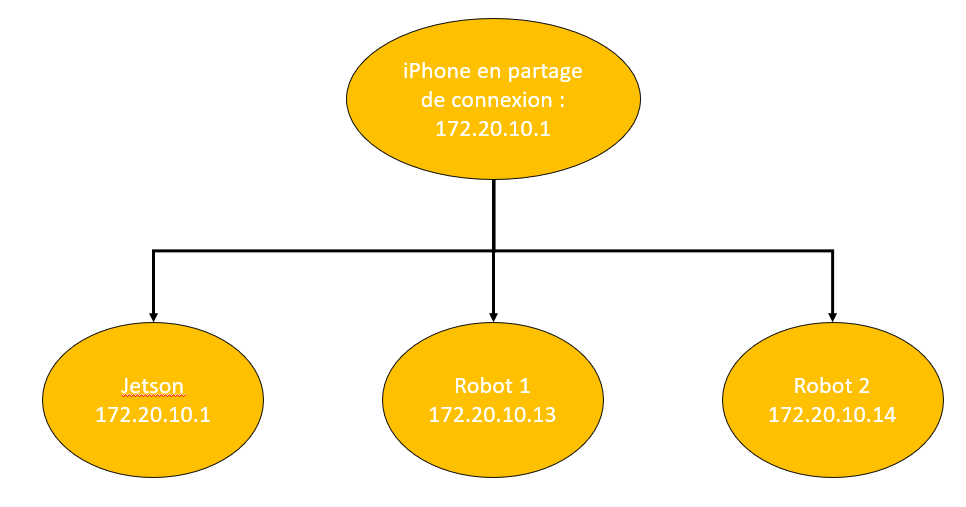
### Adaptation en cours de projet

Cependant durant le projet nous avons rencontré certains problèmes. Le plus embêtant concernait la durée nécessaire au traitement d’image pour le tracking de la balle de ping-pong. Cela serait détaillé plus tard.

Nous avons donc dû changer notre façon d’aborder les échanges de données et simplifier la chose.

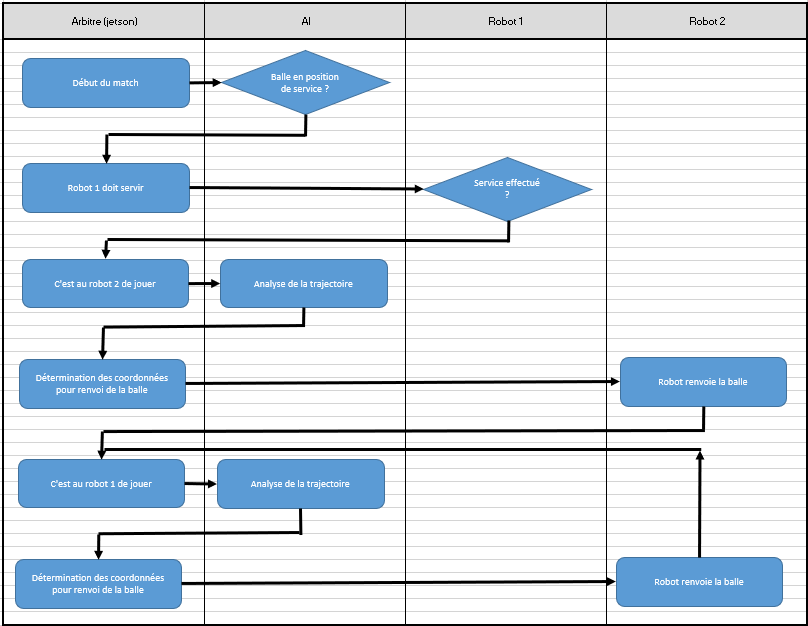
La solution que nous avons trouvé et mis en place est la suivante :

A l’aide de notre iPhone nous avons mis en place un partage de connexion. Cela revient à assimiler le téléphone à une passerelle. Le partage de connexion crée un réseau local. En y connectant nos deux robots ainsi que notre Jetson, les échanges de données étaient possibles et nous n’avions plus besoin de passer par un serveur Flask.

La mise en réseau a été relativement compliquée à mettre en place car nous ne connaissions pas la procédure. C’est pourquoi nous avons mis en annexe de ce documents les différentes captures d’écran permettant de reproduire ce réseau local. Le schéma ci-dessous reprend les différents éléments de notre réseau avec les adresses IPv4 :

## Déroulement d’un match

Avant de commencer à programmer, nous voulions savoir comment un match allait se dérouler. Pour avoir une vision claire tout au long du projet nous avons décidé de réaliser un *cross functional mapping*, qui nous permet d’avoir une vision claire de qui fait quoi.



Ainsi l’arbitre a la charge de la gestion du jeu, c’est lui qui va déterminer si le match commence ou non à condition que la balle soit en position de service. Le code python de l’arbitre va ainsi appeler un programme python qui fait servir le robot 1.

Une fois le service effectué (retour du robot à l’arbitre), l’arbitre demande à l’IA d’analyser la trajectoire de la balle. Par la suite l’arbitre va donner les coordonnées au robot 2 afin qu’il puisse renvoyer la balle.

## Commande des robots

### Mise en place des services pour chaque robot

Les coordonnées des points où les services sont effectués sont toujours les mêmes. Ainsi nous avions simplement à enregistrer les différents angles de chaque axe du robot.

Pour coder les déplacements des robots nous avons utilisé le package pyNiryo. Cette bibliothèque est très utile car elle est codée orientée objet et qui facilite grandement la programmation. Des classes et des méthodes y sont déjà implémentées et il n’y a plus qu’à les utiliser en suivant la documentation fournie.



Dans l’idée lorsque l’on veut manipuler un robot, on doit au préalable déclarer une nouvelle instance de la classe NiryoRobot en spécifiant l’adresse IP du robot (d’où l’importance de la mise en réseau des robots sur le même réseau local, afin que la jetson puisse s’y connecter).



## Arbitrage et reconnaissance d’image

### Paramétrage de la Jetson

Afin de travailler proprement et ne pas détruire l’OS de la Jetson, nous avons créer un container avec des bibliothèques de base, auxquelles nous avons ajoutées celle nécessaire pour le projet. Nous n’avons pas rebuild le container entre temps mais les fichier *DockerFile* ainsi que *requirements.txt* sont bel et bien à jour, prêt à lancer la reconnaissance d’images. La principale utilité de ce container permet d’

### Prise de vue

Afin d’être un maximum compatible avec la Jetson et l’environnement de travail (terrain de tennis à taille réduite), le cahier des charges était le suivant :

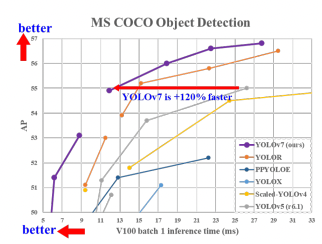
* Placement facile de la caméra donc pas trop encombrante
* Reconnue par la Jetson
* Qualité correcte
* Distorsion faible

Nous avons donc utilisé une caméra Raspberry V2.1, mais il s’est avéré que la qualité de celle-ci n’était pas adéquate : nous avions un environnement avec faible luminosité, cela ne permettait pas d’obtenir des images assez qualitatives pour détecter la balle. Un test rapide avec une prise de vue depuis un iPhone a permis de mettre en évidence ce problème car la balle a tout de suite été reconnue.

### Traitement de l’image

[](https://github.com/RizwanMunawar/yolov7-object-tracking)Comme décrit précédemment, nous avons utilisé un modèle de traitement d’image open-source nommé YOLO (version 7) : utilisé pour de nombreux projets, il est capable d’identifier les différents objets sur une image ou un flux vidéo comme l’exemple ci-dessous.

On peut donc observer que ce logiciel permet de reconnaître les objets (ici des personnes) mais surtout leur trajectoire : notre objectif est de récupérer la trajectoire de la balle afin de placer correctement le robot qui doit jouer pour frapper la balle.

[](https://blog.roboflow.com/yolov7-breakdown/)Nous avons choisi YOLO v7 parmi différents logiciels par sa rapidité d’exécution :

Après plusieurs tests, nous avons pu définir que cette solution n’est pas viable en termes de temps de traitement : 169s pour 233 images et 60s pour 94 images (vidéo de 3s à 30FPS).

De plus, la qualité de la caméra Raspberry n’est clairement pas idéale (8 mégapixels), ce qui handicape encore la reconnaissance d’image. La luminosité a aussi un grand impact sur la qualité d’image comme on peut le voir sur la photo suivante :

Une image contenant texte

Description générée automatiquement

De ce fait, la balle n’est même pas reconnue sur les différentes prises de vue effectuées.

En prenant la caméra de l’iPhone et en testant en entrée du programme de reconnaissance, on arrive bien à reconnaître la balle. De plus, ceci a été réalisé sans entraîner le model de façon particulière (nous avons pris les entraînements de base du modèle présent sur GitHub.

Nous avons mis les mêmes vidéo sur un PC (plus performant qu’une Jetson) et nous obtenons un temps de traitement d’environ 45 secondes pour 94 images, c’est donc 15 secondes de moins que le traitement via la Jetson. Nous avons tester la détection en live mais les images étaient saccadées et ne permettaient pas d’avoir un traitement et une « prédiction » de l’emplacement où taper la balle fiable et rapidement afin de pouvoir jouer au moment où la balle va arriver près du robot.

### Tracking de la balle

Pour le tracking, deux options ont été étudiées : le module SORT, qui est un complément de YOLO que l’on va nourrir d’images et va se concentrer sur l’objet qu’on souhaite tracker (dans notre cas la balle).

La deuxième possibilité serait d’utiliser Darkflow, mais le nombre d’images par secondes traitées est alors bien plus faible.

Nous n’avons pas été jusqu’à ce niveau, du fait du blocage avec la caméra qui n’est pas adaptée à la situation.

## Différents axes à suivre pour poursuivre

Etant partis de rien, nous avons surtout effectué un travail de décantage en réussissant à avoir une trame cohérente dans la manière d’aborder la programmation.

Pour les prochains étudiants qui reprendront ce projet, les différents points à améliorer ci-dessous seraient dans la continuité de notre travail :

* Amélioration au niveau de la caméra (éclairage, qualité d’image)
* Réduction du temps de traitement d’images
* Augmentation de la table (robot très lent même à 100% de leur vélocité)
* Tracking de la balle et détermination de la position de la balle : par machine learning et descente de gradient par exemple
* Gestion des fautes avec les lignes du terrain
* Envoi du flux vidéo et du score sur le serveur Flask (serveur déjà en place et fonctionnel)

## Sources

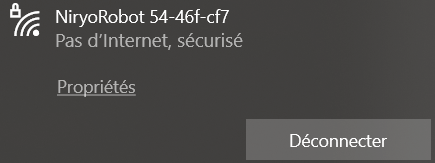
Reconnaissance d’image et tracking d’objets :

1. [Running PyTorch Models on Jetson Nano | PyTorch](https://pytorch.org/blog/running-pytorch-models-on-jetson-nano/)
2. [Pedestrian Trajectory · Issue #264 · pjreddie/darknet · GitHub](https://github.com/pjreddie/darknet/issues/264)
3. [GitHub - abewley/sort: Simple, online, and realtime tracking of multiple objects in a video sequence.](https://github.com/abewley/sort)

## Annexes

### Annexe 1 : Procédure pour mise en réseau des deux robots sur un réseau local

1. Se connecter dans un premier temps au premier robot



1. Activer votre partage de connexion

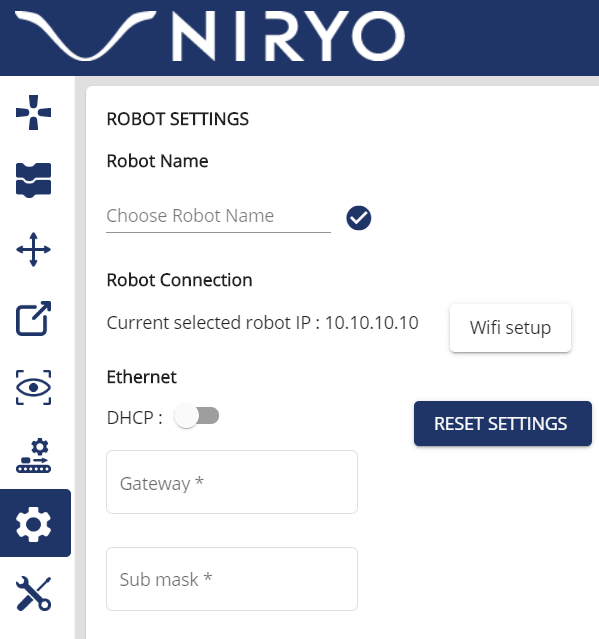


1. Dans NiryoStudio se connecter en hotspot au robot

Une image contenant texte

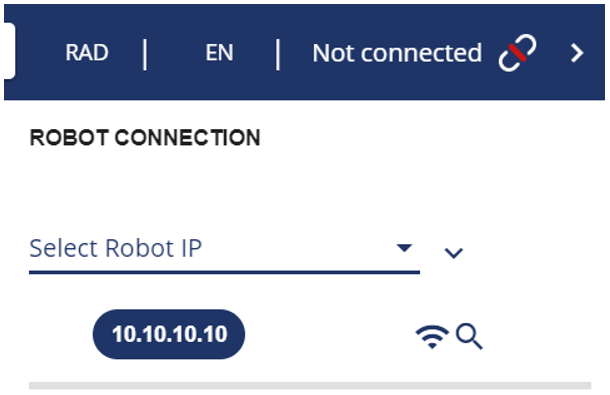
Description générée automatiquement

1. Une fois connecté en hotspot, aller dans les paramètres du robot et cliquer sur Wifi setup puis remplir les champs appropriés



Le robot est maintenant connecté à votre réseau local (partage de connexion)

1. Répéter les opérations a) c) et d) pour le second robot.
2. Connecter ensuite votre ordinateur au réseau local
3. Retourner dans NiryoStudio et recherchez les robots présents sur votre réseau



Les adresse IP des robots apparaissent dans la liste des robots disponibles, la mise en réseau a fonctionné !

Une image contenant texte

Description générée automatiquement