

Apprentissage automatique- Techniques d'apprentissage
automatique-Robotique sociale

Rapport projet pluridisciplinaire

Jordy Bacherot
Léa Mick
Cyprien Grunblatt
Naourès Haddad
M1 Sciences cognitives

INTRODUCTION	1
CONCEPTION DU SYSTÈME DE GESTES	2
Choix des gestes	2
Processus d'acquisition des données	4
ENTRAÎNEMENT ET ÉVALUATION DU MODÈLE	5
Pre-processing	5
Entraînement des modèles	6
Evaluation	8
Prédiction	8
INTÉGRATION DANS L'ENVIRONNEMENT ROBOTIQUE	9
API	9
Premier objectif : l'intégration Webots	10
Second objectif : l'intégration réelle	10
TESTS	11
CONCLUSION ET DISCUSSION	12
Optimisations	12
Inclusion	12
ANNEXES :	13

INTRODUCTION

Dans le cadre du projet pluridisciplinaire se déroulant du 2 juin au 6 juin, nous avons été amenés à mobiliser les compétences acquises au cours du second semestre, notamment en techniques d'apprentissage automatique, apprentissage automatique et robotique sociale. L'objectif final de notre travail était de concevoir un système de commande gestuelle afin de contrôler le robot humanoïde Nao dans l'environnement virtuel Webots.

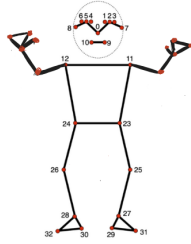
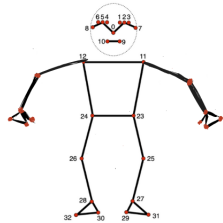
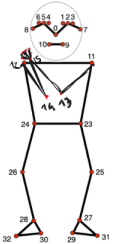
Ce projet visait à mettre en œuvre une interaction sans contact entre l'utilisateur et le robot, afin de lui permettre de contrôler les mouvements possibles du robot grâce à la reconnaissance de postures corporelles captées par la caméra de l'ordinateur puis interprétées par un modèle d'apprentissage automatique. Ce projet met à la fois en jeu des aspects techniques et humains, notamment avec la conception de gestes intuitifs, ergonomiques et accessibles par l'utilisateur. Nous nous sommes fixés pour objectif également de rendre notre système de commande inclusif, ainsi les gestes définis peuvent être réalisés avec un seul bras ou en position assise, car le contrôle de notre robot ne nécessite pas l'utilisation des jambes par l'utilisateur.

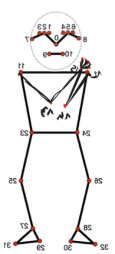
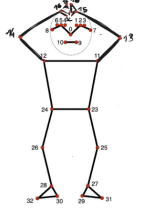
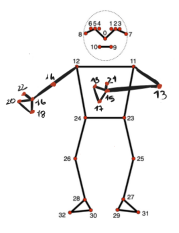
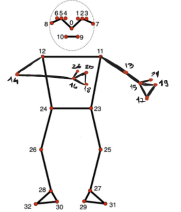
Finalement nous avons réalisés :

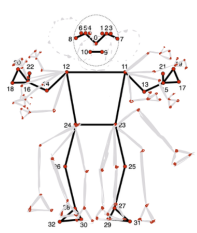
- Le système demandé par le projet, soit faire bouger le robot, à plusieurs vitesses, grâce à des gestes
- Un système permettant de bouger le robot sur WebBots à une seule main
- Un système permettant de faire bouger un vrai robot NAO (ainsi que sur chorégraphie)

CONCEPTION DU SYSTÈME DE GESTES

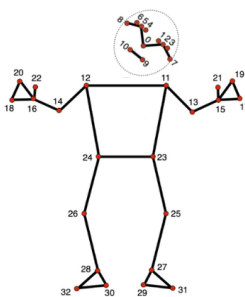
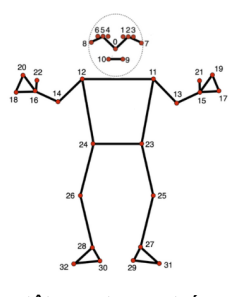
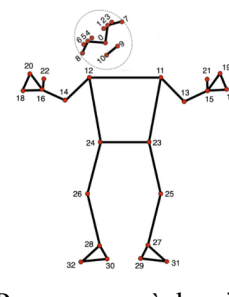
Choix des gestes

Action	Geste défini	Justification
<p>Aller vers l'avant</p>	 <p>Les bras sont pliés à 90°, vers le haut, paumes tournées vers l'avant. Le bas du corps reste fixe et orienté face à l'écran.</p>	<p>Étant donné la difficulté de représenter les mouvements d'avancer et de reculer à travers la profondeur (bras vers l'avant ou bras vers l'arrière), nous avons opté pour une représentation des gestes sur l'axe vertical. Ainsi pour avancer vers l'avant, nous avons représenté cela par les bras vers le haut. Cette posture maximise la lisibilité visuelle du geste et facilite sa détection par le modèle de reconnaissance gestuelle.</p>
<p>Aller vers l'arrière</p>	 <p>Les deux bras sont éloignés des cuisses avec les coudes légèrement pliés vers le haut. Le bas du corps reste fixe, orienté face à l'écran.</p>	<p>Ce geste est intuitif et facile à mémoriser, il évoque naturellement une intention de retrait. D'un point de vue ergonomique, il est simple à réaliser et ne présente pas d'efforts physiques importants, garantissant un confort d'usage. Sur le plan technique, il est distinctif des autres gestes et il présente une forme reconnaissable par le système de détection de geste.</p>
<p>Pivoter sur la gauche</p>	 <p>Le bras droit est replié devant le torse, positionné en diagonale entre l'épaule gauche et le bas du cou. Le bras gauche est plié vers le haut au niveau de l'épaule gauche permettant ainsi de rejoindre les deux mains, avec les doigts qui se croisent ou se touchent.</p>	<p>Le geste est facile à exécuter, et accessible même dans des espaces restreints. Il permet un effort minimal dû à la position des bras qui peuvent reposer sur la poitrine et un positionnement stable. La posture asymétrique distinctive dû au croisement des bras crée une configuration corporelle unique qui facilite la reconnaissance du geste par notre système de vision.</p>

Pivoter sur la droite	 <p>Idem que pour pivoter à gauche mais à droite.</p>	Ce geste suit la même logique que le geste qui permet le pivot gauche mais en inversant la symétrie. Ainsi les mains se croisent entre l'épaule droite et le bas du cou.
Saluer	 <p>Les deux bras sont levés au niveau du front avec les coudes légèrement fléchis et paumes de mains tournées vers l'avant. Le bas du corps reste fixe, orienté face à l'écran.</p>	Comme il ne nous restait que peu de choix de gestes suffisamment distinctifs des autres, nous avons opté pour ce geste pour faire saluer le robot. Ce geste a été choisi pour sa simplicité et sa possibilité d'être réalisé par un bras étant donné que c'est un geste symétrique.
Faire un pas sur la gauche	 <p>Le bras gauche est tendu latéralement vers la gauche, le bras droit est plié devant la poitrine orienté vers la gauche, paume visible. Le bas du corps reste fixe, orienté face à l'écran.</p>	Ce geste s'inspire d'un geste instinctif pour désigner une direction dans un contexte social. La posture combine une asymétrie marquée (un bras tendu, un bras plié) et une orientation définie, ce qui rend le geste facilement reconnaissable par le modèle. Il est ergonomique et accessible car il ne demande pas de déplacement des jambes.
Faire un pas sur la droite	 <p>Idem que pour faire un pas à gauche mais à droite.</p>	Pour les mêmes raisons que "faire un pas à gauche" nous avons choisi le même geste mais à droite.

Rien	 <p>Tous les gestes non définis parmi les gestes précédents sont considérés comme des gestes d'inaction, le robot ne fait rien.</p>	X
------	--	---

Les gestes ont été réfléchis de manière à pouvoir aussi être réalisés à un seul bras, tout en gardant des résultats satisfaisants..

Vitesse 1	Vitesse 2	Vitesse 3
 <p>La tête est inclinée vers la droite pour mettre le robot en vitesse lente</p>	 <p>La tête reste centrée dans l'axe du corps pour indiquer l'intention de l'utilisateur à mettre le robot en vitesse intermédiaire</p>	 <p>Pour passer à la vitesse rapide, l'utilisateur incline sa tête vers la gauche.</p>

L'indication de l'intention de changement de vitesse grâce à l'inclinaison de la tête était pour nous un choix naturel, car elle permet à l'utilisateur de garder les mains libres. Il est donc possible d'indiquer le mouvement souhaité du robot à l'aide des bras, tandis que la vitesse est contrôlée indépendamment par la tête, afin d'éviter la confusion possible entre le mouvement et l'intensité du mouvement. Afin de permettre une expérience plus agréable à l'utilisateur, nous avons attribué la vitesse intermédiaire à la tête centrée, afin que le robot démarre automatiquement à cette vitesse, que nous estimons être la plus confortable (ni trop lent, ni trop rapide).

Nous avons donc 2 traitements de données, un pour la vitesse (avec la tête) et l'autre pour le mouvement (avec la buste, les bras et la tête) et donc 2 modèles.

Processus d'acquisition des données

Afin de constituer un jeu d'exemples variés et représentatifs pour entraîner notre modèle de reconnaissance de gestes, nous avons procédé à la phase d'acquisition des données grâce à différents participants. Dans un premier temps chaque membre du groupe a réalisé dix fois chaque gestes définis, en introduisant des variations dans l'exécution (distance par rapport à la caméra, amplitude du mouvement...). Puis pour enrichir les données, nous avons également sollicité la participation d'autres étudiants et enseignants, il leur a été demandé à chacun d'eux de reproduire les gestes, également dix fois chacun, tout en appliquant des variations. Nous avons aussi changé de pièces et de zones de prises de données pour ajouter le maximum de variation dans les données.

ENTRAÎNEMENT ET ÉVALUATION DU MODÈLE

Cette section fait partie du dossier IA du projet. Les étapes de traitement y sont détaillées, avec une indication des fichiers dans lesquels elles sont implémentées, ce qui permet de suivre l'enchaînement des processus.

Vous pouvez retrouver en annexe :

- Le lien vers le projet de Github
- La structure du projet (que nous vous conseillons d'aller voir pour une compréhension plus simple de la suite)

Pre-processing

Notre preprocessing se compose de différents fichiers python, contenant des fonctions dédiées à la préparation des données. Ces fonctions se trouvent dans le dossier `exploration_data`.

Nous avons un premier fichier "`concat.py`", qui contient les différentes fonctions de concaténation des fichiers. Nous avons ensuite le fichier "`preprocessing.py`", qui contient les fonctions de normalisation, les fonctions permettant de détecter les outliers, la data augmentation, les fonctions permettant de créer le modèle pca, ainsi que la fonction pour créer la matrice de corrélation. Enfin, nous avons le fichier "`data_viz.py`" qui contient les différentes fonctions de data visualisation. Cela comprend la heatmap pour la matrice de corrélation, les boxplots, et les visualisations pour les PCA (matrice de nuages de points, visualisation 2D et 3D des features et des points).

Toutes ces fonctions sont utilisées dans les notebooks “`data_preprocessing.ipynb`”, et “`data_preprocessing_one_arm.ipynb`”. Les données sont chargées depuis une des fonctions de la classe `concat` puis séparée pour la vitesse et pour le mouvement du robot. Les données subissent ensuite une data augmentation et une normalisation, spécifique au problème, avant d’afficher les boxplots et la heatmap pour les matrices de corrélations. Nous pouvons ensuite effectuer le PCA pour réduire le nombre de features. Nous avons pu interpréter les différentes composantes, le nombre optimal de composants étant donné par la méthode du coude. Nous avons pu visualiser les PCA en 3D dans le cas où il y avait 3 composantes, et en 2D lorsqu’il n’y en avait que 2, avec les différents points représentant les individus, et les features.

Des explications plus poussées se trouvent dans les notebooks cités ci-dessus.

Entraînement des modèles

Une fois les données traitées, elles sont enregistrées dans les fichiers suivants :

- “`preprocessing_data`” : contient les données destinées aux modèles de classification des mouvements et des vitesses, ainsi que les features correspondant aux besoins du projet, ou leur réduction via une analyse en composantes principales (PCA).
- “`preprocessing_data_one_arm`” : contient les données spécifiques au modèle de classification des mouvements d’un seul bras.

À partir de ces fichiers, nous avons ensuite porté notre attention sur les modèles.

Je vais désormais présenter notre travail sur les modèles dans l’ordre chronologique des étapes effectuées durant la semaine. Comme dit plus tôt, les modèles ont évolué par des changements dans les données durant la semaine, notamment un changement de gestes.

L’objectif initial était d’explorer rapidement plusieurs modèles afin d’identifier les plus prometteurs, puis d’approfondir leur performance en ajustant leurs hyperparamètres.

Cependant, certains modèles que nous pensions n'utiliser que de manière exploratoire (en particulier le Random Forest que l’on a sélectionné) ont donné des résultats si satisfaisants que nous n’avons pas jugé nécessaire de les optimiser davantage.

Ces expérimentations ont été réalisées dans le dossier “`model_exploration`”, à l’aide de plusieurs notebooks nommés “`model_exploration_....ipynb`”, chacun dédié à un modèle spécifique.

Nous avons également utilisé la classe “`explore_models.py`” pour centraliser certaines fonctions d’exploration.

Dans les dossiers de IA, on retrouve les modèles suivants (les points représentent l’ordre chronologique) :

1. Sauvegardés dans le dossier “**models_temp**” (modèles RandomForest) :
 - un modèle de classification de mouvement à deux bras
 - un modèle de classification de mouvement à deux bras avec les données PCA (non utilisé car plus mauvais que le modèle classique)
 - un modèle de classification de vitesse
 - un modèle de classification de vitesse avec les données PCA (non utilisé car plus mauvais que le modèle classique)
 - les labels encoders qui seront réutilisés pour la prédiction
2. Sauvegardés dans le dossier “**models_ANN**” (étape que j'expliquerais plus en détail plus bas) :
 - un modèle de classification mouvement à deux bras (ANN)
 - un label encoder qui sera réutilisé pour la prédiction
3. Sauvegardés dans le dossier “**models_one_arm**” (Random Forest)
 - un modèle de classification mouvement à un bras
 - un label encoder qui sera réutilisé pour la prédiction

Après avoir sélectionné les quatre premiers modèles, nous avons souhaité aller plus loin en explorant une architecture de type réseau de neurones (ANN) à l’aide de Keras pour la classification des déplacements.

Bien que les performances des modèles Random Forest aient été excellentes, l’exploration d’un réseau de neurones (ANN) nous a offert un avantage supplémentaire : la possibilité de manipuler directement les probabilités de sortie. Cette flexibilité nous a permis de définir deux seuils distincts, dans le but de renforcer la prédiction de la classe "neutre" et ainsi limiter les mouvements involontaires lors de l’inférence, ces seuils seront réutilisés dans la prédiction.

Pour optimiser ce modèle, nous avons procédé en trois étapes :

- Une recherche aléatoire (RandomSearch) pour explorer efficacement l’espace des hyperparamètres.
- Un affinage via GridSearch autour des meilleures configurations identifiées.

- Une analyse du modèle final, incluant la validation sur le jeu de validation.
- L'exploration des différents seuils pour trouver les meilleurs.

Cette expérimentation est documentée dans le notebook “`explore_ANN_model_move.ipynb`”. Attention aux versions des dépendances “`numpy`” et “`tensorflow`”.

Nous n'avons pas réalisé d'ANN pour la prédiction des vitesses, car augmenter le “neutre” de la vitesse, c'est-à-dire “moyen” avait moins de sens et d'utilité.

Enfin, nous avons réalisé un modèle spécifique à un seul bras. Comme indiqué précédemment, il repose sur les mêmes principes que les autres modèles, mais n'utilise pas les données du bras gauche (le bras gauche ayant été supprimé lors du prétraitement). Un nouveau préprocessing a été effectué, suivi de l'entraînement, et une fois encore, le modèle Random Forest s'est avéré le plus performant.

Cette partie est également décrite dans le notebook “`explore_ANN_model_move.ipynb`”.

Evaluation

Compte tenu des très bons résultats obtenus par les différents modèles, nous n'avons pas approfondi l'évaluation au-delà des indicateurs de base.

L'analyse s'est principalement concentrée sur l'accuracy ainsi que sur les matrices de confusion des meilleurs modèles. Ces dernières ont révélé que les rares erreurs de classification se concentrent principalement sur la classe “neutre”, ce qui est cohérent avec son rôle de classe intermédiaire ou ambiguë dans le contexte du projet.

Prédiction

La prédiction à l'aide des différents modèles est gérée via une API développée avec FastAPI en Python.

Quatre classes principales sont exposées pour interagir avec les modèles, leur intégration sera détaillé plus bas dans le rapport (partie Webots) :

- “`model_API.py`”, l'API commune :
 - Utilisant “`best_model_move.pkl`” du dossier `models_temp`
 - Utilisant “`best_model_speed.pkl`” du dossier `models_temp`
 - Les labels encoders associés
- “`model_API_ANN.py`”, l'API pour utiliser l'ANN :

- Utilisant “best_ann_model_move.h5” du dossier models_ANN
- Utilisant “best_model_speed.pkl” du dossier models_temp
- les labels encoders associés
- “model_API_one_arm.py”, l’API pour utiliser sans le bras gauche :
 - Utilisant “best_model_one_arm_move.pkl” du dossier models_one_arm
 - Utilisant “best_model_speed.pkl” du dossier models_temp
 - les labels encoders associés
- “model_API_mediapipe.py”, cette API est particulière car mediapipe est coté API et non plus côté client, ce qui permet d’utiliser un robot Nao directement et non plus la simulation WeBots. (Elle ne peut pas être utilisé correctement sans le robot NAO ou choregraph)
 - Utilisant “best_model_move.pkl” du dossier models_temp
 - Utilisant “best_model_speed.pkl” du dossier models_temp
 - Les labels encoders associés
 - Intégrant médiapipe et ses processus

À l’intérieur de chaque API, on retrouve une fonction “predict” chargée de préparer les données puis de transmettre la prédiction. Cette fonction applique les transformations nécessaires, notamment :

- Reçoit les données
- La normalisation des données d’entrée,
- La prédiction des modèles
- Dans le cas du modèle ANN, l’application d’un seuil ajusté pour renforcer la détection de la classe “neutre”.
- L’utilisation des LabelEncoder associé aux modèles, afin de convertir les indices de sortie en classes prédites
- Retourner les classes prédites

INTÉGRATION DANS L’ENVIRONNEMENT ROBOTIQUE

Après avoir travaillé sur le traitement de nos données et nos différents modèles. Nous avons choisi d’intégrer notre interprétation Mediapipe via une API de type FastAPI.

API

Nous avons choisi d'utiliser une API pour différencier les calculs de notre application Webots des

calculs du modèle et aussi permettre de faire fonctionner ces tâches sur différents supports. Cette décision se justifie par notre volonté d'éviter de surcharger l'application et par la difficulté d'intégrer directement Mediapipe dans notre robot Nao. Ce dernier fonctionne sous Python 2.7, ce qui complexifierait une intégration directe de Mediapipe.

Tuto d'utilisation :

- Aller dans le dossier IA comportant le script `model_API.py` ou `model_API_onearm.py`
- Lancer dans le terminal la commande : `fastapi dev model_API_one_arm.py`
- Lancer le projet webots une fois l'API bien démarré (en connectant l'url de l'API dans le fichier "pose.py")

L'utilisation d'une FastAPI nous permet de remplir deux objectifs.

Premier objectif : l'intégration Webots

Naturellement, notre priorité initiale était d'intégrer notre projet dans Webots. Pour ce faire, nous avons adapté et intégré les divers scripts fournis pour qu'ils correspondent à nos besoins spécifiques. En premier lieu, le script "**pose.py**" a été modifié pour permettre l'envoi de requêtes POST contenant les données de Mediapipe à notre API. Ensuite, nous recevons un tableau indiquant la vitesse et le mouvement requis, basé sur l'interprétation de notre modèle.

Une fois ce tableau reçu, nous l'utilisons pour commander les mouvements du robot. Cette méthode est également employée dans le script "**nao_pp_s8.py**" où le robot réagit aux instructions reçues. De plus, nous avons ajusté la gestion de la vitesse pour permettre à l'utilisateur d'augmenter ou de réduire la vitesse en inclinant la tête. Enfin, des améliorations logiques ont été ajoutées au code dans la partie "**naomotion.py**".

- Des conditions qui contrôlent les arrêts de nos mouvements pour éviter de superposer les différentes animations,
- Une fonction `_wait` qui permet d'ajouter des temps arrêts afin de permettre un meilleur rythme entre les animations et les requêtes API,
- Une animation coucou qui permet de lancer une motion particulière pour créer une animation "surprise",
- Une fonction `cant_move` qui bloque les mouvements si le robot est face à un mur.

Trois nouveaux nœuds ont été ajoutés au robot, spécifiquement au niveau du bodySlot et du headSlot. Premièrement, un capteur de distance sonar, simulant celui du robot Nao, a été intégré au bodySlot pour détecter les objets entre 0.25 et 2.25 mètres. Ensuite, un haut-parleur a été

ajouté à la tête du robot, lui permettant de communiquer verbalement, comme signaler les changements de vitesse, saluer ou avertir de la présence d'un mur. Enfin, une caméra offrant une vue à la troisième personne a été installée pour faciliter le contrôle du robot lors de la réalisation du parcours de manière plus ergonomique. C'est pour cela que les commandes sont inversées dans le script Webots. Deux derniers points d'ergonomie, il y a une régulation automatique des vitesses lorsque que l'on demande au robot de faire un pas de côté. En effet, nous forçons le robot à passer en vitesse deux car il pourrait tomber s'il reste en vitesse trois. Le second point consiste à, si le modèle ne détecte plus de corps, alors le script envoie automatiquement une position neutre à vitesse moyenne pour éviter les crashes d'absence de direction.

Second objectif : l'intégration réelle

L'étape suivante de notre projet consistait à piloter un véritable robot Nao à l'aide de notre script. Pour y parvenir, nous avons adapté notre script initial, `model_API.py`, en `model_API_mediapipe.py`. Ce nouveau script intègre Mediapipe pour l'interprétation des gestes et expose les résultats via une méthode GET. Afin de communiquer avec notre API, nous avons ajouté manuellement la bibliothèque `requests` à notre installation Python 2.7. Pour la programmation des mouvements du robot, nous avons exploité les fonctions existantes de *naoqi*, en particulier celles de *ALMotion*. Nous avons notamment utilisé la fonction *moveTo*, qui permet de déplacer le robot selon une distance et/ou un angle (exprimé en radians) spécifiés.

Le script modifié est fonctionnel et nous offre la possibilité de commander un véritable robot Nao par le biais de nos gestes. Celui-ci est perfectible, mais dans le but de respecter le temps imparti, nous ne pousserons pas plus loin son développement.



[Voir la vidéo jointe en annexe.](#)

TESTS

Une fois le modèle prêt et le projet Webots mise en place, nous avons fait tester les gestes définis individuellement par différentes personnes. Les résultats étaient globalement satisfaisants mais nous avons pu constater quelques erreurs d'interprétations entre deux de nos gestes.

En effet, notre modèle confondait parfois les gestes que nous avons initialement définis comme “pivoter” et les gestes “pas sur le côté”. Nous avons donc décidé de modifier les gestes de “pivoter” afin d’éviter cette confusion, mais aussi pour faciliter notre modèle à un bras dans le but de rendre celui-ci plus inclusif. Ces choix de positions maintenant plus asymétriques et distinctes ont permis d’améliorer la précision de notre modèle.

Lors de nos essais, maintenir la tête pivotée pour atteindre une certaine vitesse s'est avéré inconfortable. Par conséquent, nous avons opté pour un système d'incrémentement et de décrémentement de vitesse. Cette solution permet de pencher la tête uniquement lors d'un mouvement spécifique pour ajuster la vitesse, offrant ainsi une meilleure expérience utilisateur.

Nous avons continué à effectuer des tests, que ce soit sur un véritable robot NAO, ainsi que pour notre modèle à un bras et notre modèle de base, ce qui nous a permis de constater que malgré quelques confusions avec les positions neutres, notre modèle fonctionnait de manière satisfaisante, et le robot suivait bien les instructions.

CONCLUSION ET DISCUSSION

Nous vous conseillons de favoriser l'utilisation de l'API "model_API.py" ou "model_API_one_arm.py", contrairement à "model_API_ANN.py", puisqu'elles n'ont pas de dépendance avec Numpy et TensorFlow.

Optimisations

Plusieurs points dans notre projet mériteraient davantage de travail, qui pourrait permettre d'atteindre de meilleurs résultats :

- Utilisation de Pipeline pour simplifier les processus d'entraînement et la réutilisation dans la prédiction (éviter de tout refaire à la main)
- Retirer des colonnes avec les résultats des PCA (éléments du visage et des épaules pour la prédiction des mouvements du robot, et épaules pour la prédiction de la vitesse)
- Optimiser et structurer les fichiers et le code
- Possible d'étudier le ANN avec ridge, lasso et elastic search pour voir si les résultats seraient plus intéressants.
- Améliorer les modèles gérant la prédiction de la vitesse, les données d'entraînement actuelles étant de mauvaises qualités, rendant les prédictions moins fiables.
- Idée : Réfléchir à un moyen d'intégrer une moyenne des cinq mouvements précédents plutôt que d'un seul mouvement, pour éviter la prise de données lors d'une transition non souhaitée.

Inclusion

L'inclusion soulève plusieurs interrogations, notamment sur la manière de contrôler notre robot sans recourir à des mouvements moteurs. L'option de la commande vocale, via l'API ALDialog de Nao pour la reconnaissance vocale, se présente comme une solution simple. L'intégration d'un modèle Speech to text sur Webots serait nécessaire. De plus, la programmation d'un modèle de reconnaissance faciale pour piloter le robot avec le visage est envisageable. Diverses pistes intéressantes et explorables s'offrent à nous, mais nous nous sommes arrêtés ici.

ANNEXES :

- Structure du projet :

```
projet_pluri_2/
├── IA/          # Tous les scripts concernant le traitement des données et les modèles en python
├── data/        # Ensemble des données enregistrées via mediapipe et regroupé par
                # action dans des dossiers
├── data_regrouped_unprocessed/  # Données regroupées non traitées pour
                                # l'entraînement des modèles de classification de mouvements et de vitesses
├── exploration_data/           # Fonction python pour explorer les données utilisé dans les
                                # notebooks python pre_processing_data et one_arm
├── model_exploration/          # Contient les notebooks d'exploration et de sauvegarde
                                # des modèles
├── model_ANN/                  # Contient les modèles de classification de mouvements (Keras)
├── model_one_arm/              # Contient le modèle de classification de mouvements pour
                                # un bras (Keras)
├── model_temp/                 # Contient les modèles de classification de mouvements et des
                                # vitesses (RandomForest)
├── preprocessing_data/         # Données prétraitées pour l'entraînement des modèles de
                                # classification de mouvements et de vitesses
├── preprocessing_one_arm_data/ # Données prétraitées pour l'entraînement du
                                # modèle de classification de mouvement pour un bras
├── data_preprocessing_....ipynb # Notebooks pour le prétraitement des données
├── model_API_....py            # Différents scripts pour les API FastAPI
├── partie_choregraph/         # Contient les scripts python 2.7 à exécuter sur choregraph
├── partie_webots/             # Comportements/flows pour NAO via Choregraphe
└── README.md                  # Readme du projet
```

- Lien GitHub : https://github.com/Neirpy/projet_pluri_2
- Lien vidéo : https://drive.google.com/file/d/1iKLmaX9GtuSC1XMZ3r-9si0drpcUX77j/view?usp=drive_link