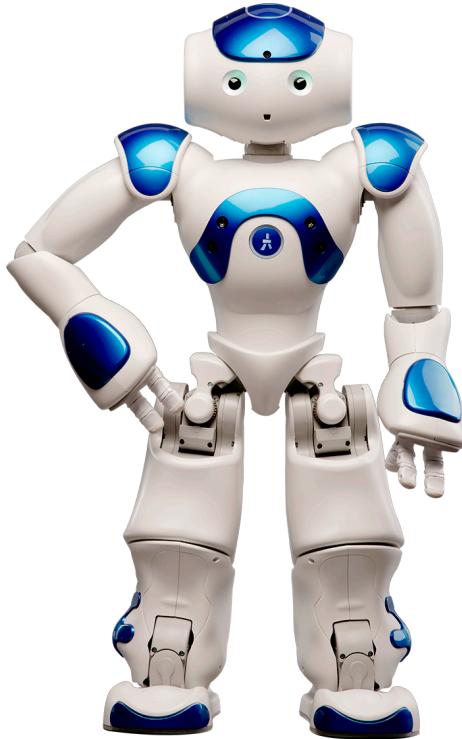


Projet NAO

Rapport final



Intervenants

Alain VISSAULT, Nicolas FEREY

Chef de projet

Gabriel GAUTIER

Adjoint

Loïck GOURVIL

Membres

**PIRIS Marius, LIM Remy, PHAM Léo, STEVANOVIC Lazar,
CORDIER Moïra, DUBOCQ Keryann, LI Haolong, JIN Tiancheng**

Sommaire :

Introduction	4
Définition du projet	5
Contexte	5
Objectifs	5
Expression du besoin	5
Description détaillé du projet	6
Cas d'utilisation	6
Le diagramme fonctionnel	7
Organigramme technique	7
Maquettes	8
Page de connexion et d'accueil	9
Le mode contrôle	9
Le mode autonome	10
Livrables	10
Organisation du groupe	11
Outils d'organisation	13
Liste des tâches techniques	15
Description des tâches techniques	18
Liste des tâches	18
1. Téléphone	18
1. 1. Connexion réseau	18
1. 2. Gestion de flux (téléphone)	18
1. 3. Réception et envoi audio	19
1. 4. Gestion de flux vidéo en direct entre le téléphone et Nao	19
1. 5. Géolocalisation	20
1. 6. Gyroscope	20
1. 7. Livrable - Maquette	21
1. 8. Livrable – Développement interface	22
1. 9. Livrable – Gestion des paramètres	22
2. Nao	24
2. 1. Mouvement tête	24
2. 2. Mouvement jambes / bras	24
2. 3. Gestion déplacement jambes / bras	25
3. Interprétation	26
3. 1. Reconnaissance vocale	26
3. 2. Traitement des ordres	26
3. 3. TTS (Text To Speech)	27
3. 4. Reconnaissance d'objet	27
GANTT	29
Développement du Nao	30
Répartition des tâches techniques	30

Liste des outils de programmation	32
Réalisation	34
Organisation du dépôt git	34
Mise en place du projet Android Studio	34
Initialisation du robot Nao	34
Mise en place du projet Android Studio	35
Débogage du projet Android Studio	36
Programmation des interfaces de l'application sur Android Studio	36
Récupération et envoi des données du gyroscope	38
Récupération et envoi des données de géolocalisation	44
Réception et envoi audio à partir du Téléphone	50
Tâche Gestion du flux vidéo en direct entre le téléphone et le robot	51
Programmation du TextToSpeech	54
Réception et envoi de l'audio à partir du robot NAO	55
Gestion du déplacement des mains	56
Gestion du déplacement des Pieds	59
Tester les mouvements du NAO	63
Intégration	67
Intégration des code du gyroscope, de la géolocalisation et du SpeechToText	67
Intégration des codes Eclipse au projet Android Studio	67
Intégration des jambes, tête et mains pour la gestion du déplacement global	68
Intégration dans le serveurs python du traitement des mouvements	72
Conclusion	74

Introduction

Objectif du Projet :

L'objectif de notre groupe est de créer un projet nommé "AVATAR NAO". Ce projet vise à permettre à l'utilisateur de contrôler le robot NAO de manière similaire au film AVATAR. L'utilisateur utilisera un casque de réalité virtuelle (VR) pour voir ce que NAO voit, et lorsque l'utilisateur bouge la tête, NAO imitera ce mouvement. De plus, lorsque l'utilisateur se déplace, NAO se déplacera également en conséquence. Un autre objectif est de permettre à l'utilisateur de donner des commandes vocales à NAO, par exemple, dire "assieds-toi" et NAO s'assiéra. Le choix des commandes vocales est laissé à la libre sélection de notre équipe.

Composition de l'Équipe :

Notre équipe se compose de 10 membres, chacun ayant des tâches et des objectifs spécifiques à accomplir dans le cadre du projet. L'équipe est structurée de manière à tirer parti des compétences et des connaissances de chacun de ses membres pour atteindre les objectifs du projet.

Deux Groupes avec des Approches Différentes :

Le projet comporte deux groupes distincts, chacun ayant sa propre organisation, gestion et approche des problèmes. Les deux groupes travaillent sur le même projet, mais ils abordent les défis de manière différente. Cette approche concurrentielle favorise l'innovation et la créativité, tout en permettant aux deux groupes de se mesurer l'un à l'autre dans une compétition amicale.

Chaque groupe est responsable de créer une version fonctionnelle du projet AVATAR NAO selon sa propre stratégie et ses priorités. Les deux groupes cherchent à surpasser l'autre en termes de performance, d'efficacité et de créativité

Cahier des charges

Définition du projet

Contexte

Le projet S5 consiste à travailler en équipe plus conséquente qu'auparavant sur des thèmes complexes et variés (Robot, montre connecté, jeu Unity). Le sujet de notre classe A est celui du robot NAO. La classe a été divisée en deux groupes. Ce cahier des charges aura pour but de présenter le projet avec ses besoins et ses livrables.

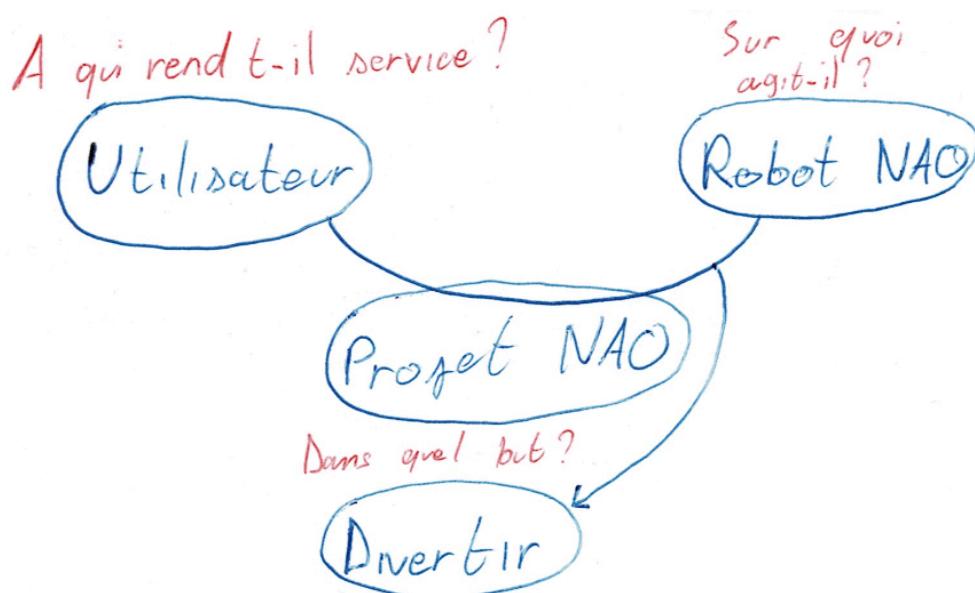
Objectifs

Pour le projet S5, le but sera de créer une application Android afin de contrôler le robot NAO à distance. Ce projet se fera avec 10 personnes.

Aucun public n'est visé pour ce projet. Le but principal est d'initier les étudiants de S5 à la gestion de projet avec des grosses équipes. Mais ce projet peut potentiellement servir pour des opérations à distance sur des lieux inaccessibles, à risques.

Expression du besoin

Le projet Robot NAO a pour but d'ajouter des fonctionnalités au robot et permettre d'interagir avec à l'aide d'un téléphone Android. Il faudra notamment pouvoir contrôler le robot en utilisant le téléphone comme un casque de réalité augmentée. Voici le diagramme bête à cornes du projet NAO :



Voici les besoins que le projet devra remplir au minimum :

- Intégrer 2 modes de déplacement pour le robot (Mode contrôle et autonome):
 - Mode contrôle: L'utilisateur pourra contrôler le robot avec un casque VR.
 - Mode autonome: L'utilisateur pourra donner des ordres au robot.
- Pouvoir dialoguer avec le robot (Discuter, Donner des ordres).
- Intégrer de la reconnaissance d'objets et d'obstacles pour le robot.

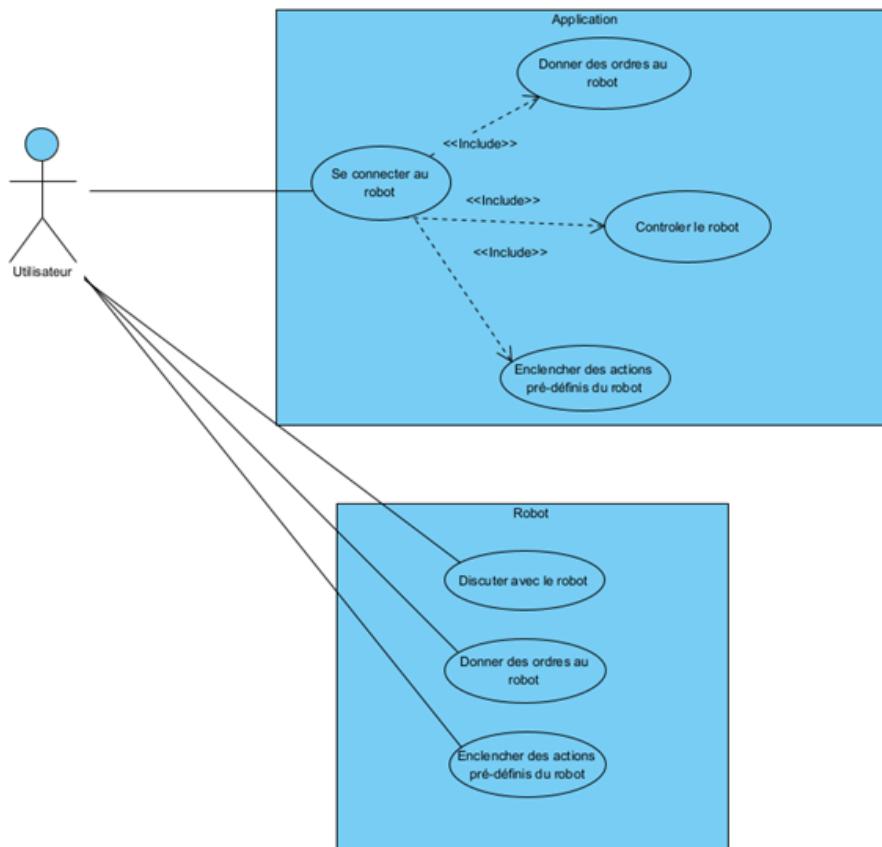
Voici les besoins que le projet pourra remplir si les besoins principaux ont été intégrés:

- Intégrer des gestuelles de courtoisie
- Intégrer des danses
- Faire des échanges musicaux et textuels

Description détaillé du projet

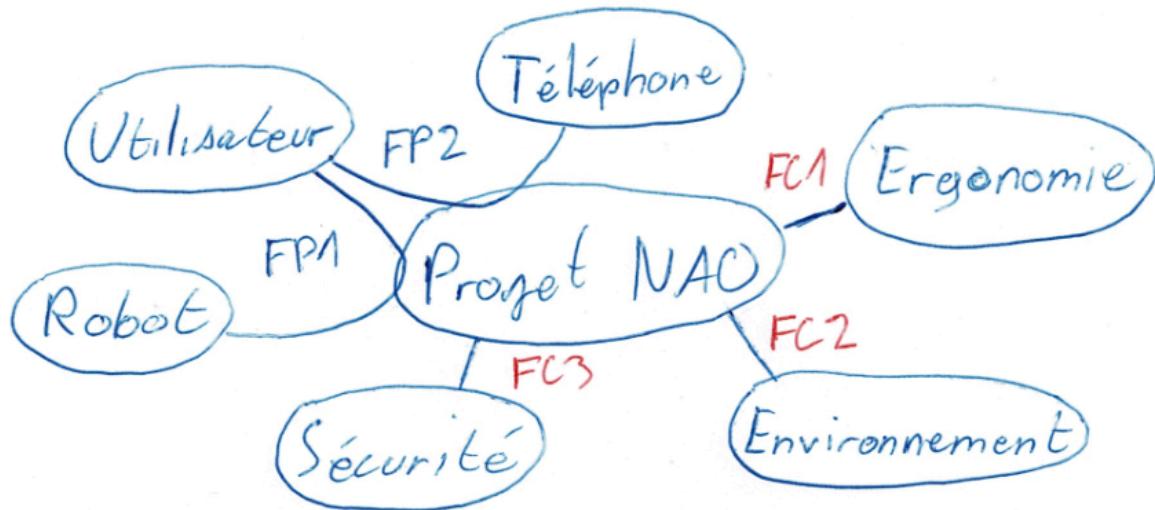
Cas d'utilisation

Voici les diagrammes de cas d'utilisation qui montrent les différentes interactions de l'utilisateur avec le robot et le téléphone :



Le diagramme fonctionnel

Voici le diagramme pieuvre qui résume l'ensemble des fonctions principales, complémentaires et contraintes du système et des éléments du milieu extérieur :

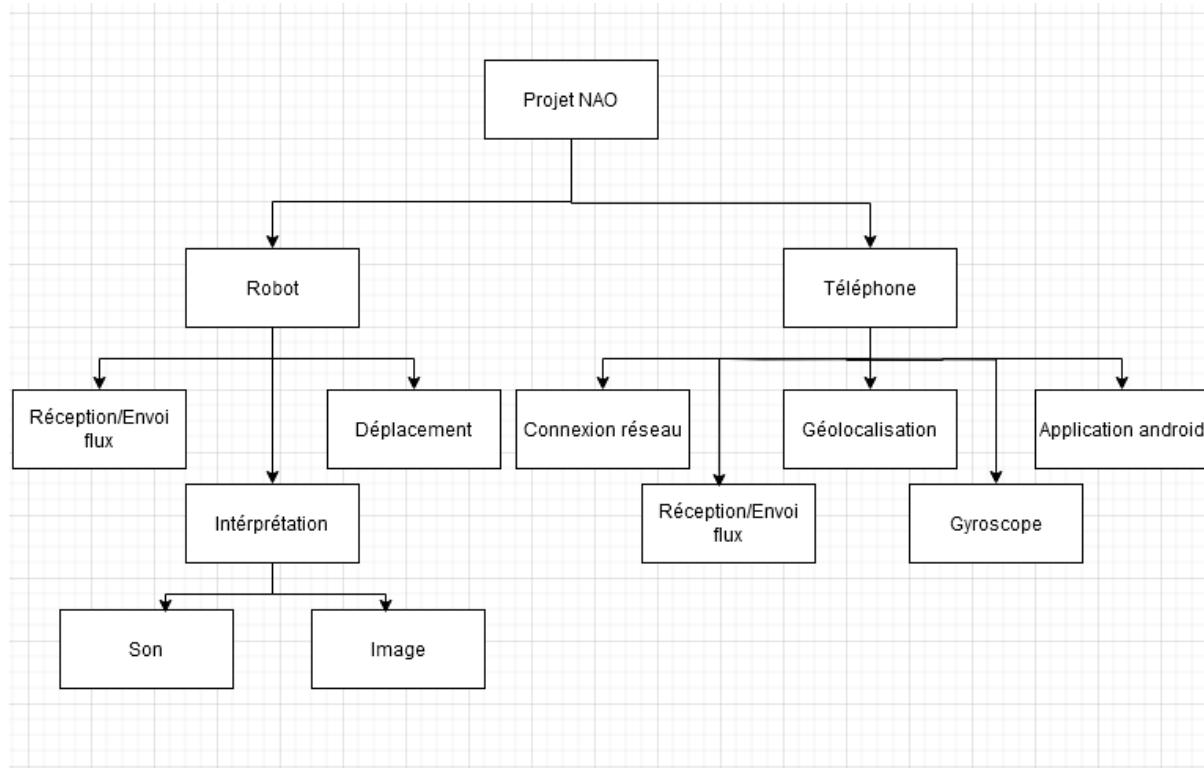


Voici le tableau qui liste les fonctions du diagramme avec leur importance de 1 (le plus important) à 3 (le moins important) :

Code	Fonction	Importance
FP1	<ul style="list-style-type: none"> Discuter avec le robot Donner des ordres au robot 	1
FP2	<ul style="list-style-type: none"> Donner des ordres au robot Faire des actions pré-définis Contrôler le robot avec le téléphone 	1
FC1	<ul style="list-style-type: none"> Naviguer facilement dans l'application Utiliser simplement l'application 	3
FC2	<ul style="list-style-type: none"> Repérer les obstacles Reconnaître des objets Reconnaître l'utilisateur 	4
FC3	<ul style="list-style-type: none"> Utiliser le robot sans l'endommager 	2

Organigramme technique

Voici l'organigramme technique qui découpe le projet en sous-catégories :



Pour le projet NAO, on distingue deux branches pour le projet : celle qui s'occupera des tâches techniques liées au robot et celle au téléphone.

Pour le robot, il faudra implémenter :

- la réception de flux audio et des coordonnées et l'envoi de flux vidéo et audio qui vont permettre de passer les données au téléphone ou les récupérer.
- l'interprétation des données audio et vidéo pour la reconnaissance vocale, le traitement de ordres et la détection d'objets, d'obstacles.
- les déplacements du robot comme tourner la tête, avancer, reculer, bouger les bras.

Pour le téléphone, on devra intégrer :

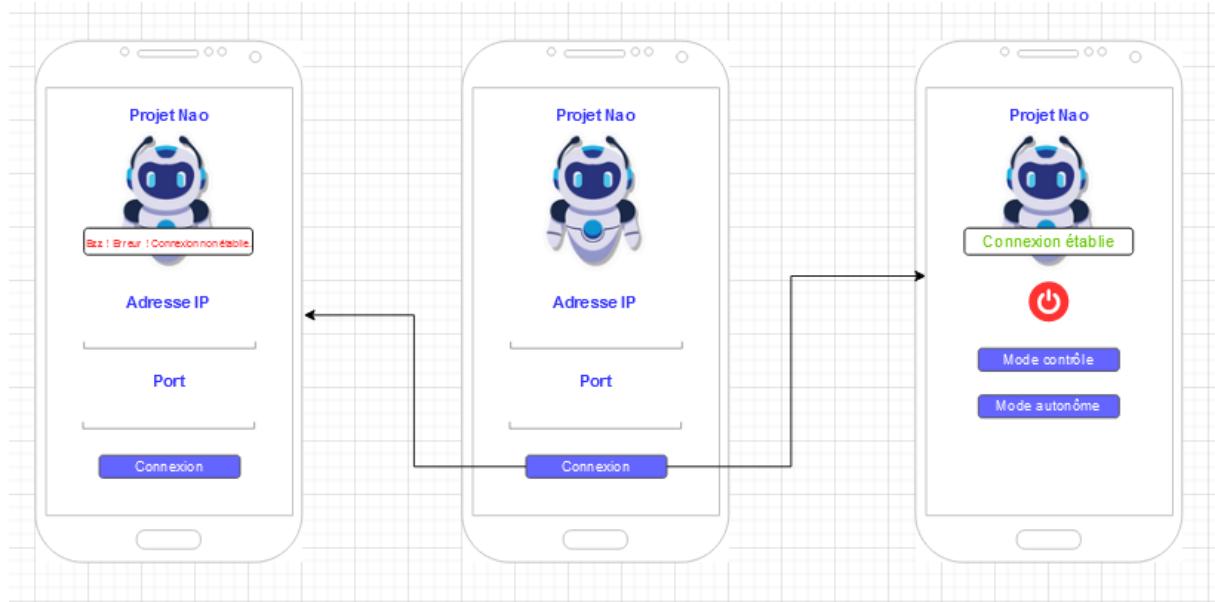
- la connexion réseau pour se connecter au robot.
- la réception de flux vidéo et audio et l'envoi de flux audio et des coordonnées.
- la géolocalisation du téléphone pour déplacer le robot avec le téléphone.
- le gyroscope pour tourner la tête du robot.
- l'application android servant d'interface pour l'utilisateur.

Cet organigramme permettra d'établir une liste des tâches qui sera la plus détaillée possible.

Maquettes

Voici les maquettes de l'application Android:

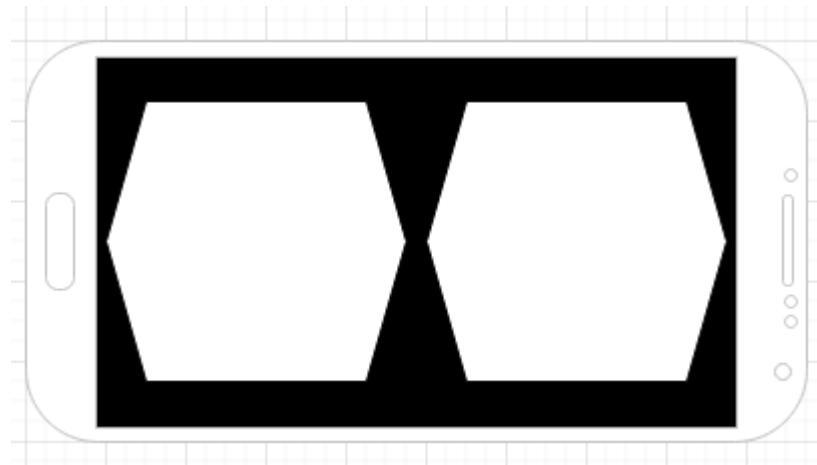
Page de connexion et d'accueil



Pour pouvoir interagir sur le robot avec le téléphone, l'utilisateur devra se connecter au robot en saisissant son adresse ip et son port. Si la connexion réussit, l'utilisateur sera notifié de la réussite de la connexion et aura accès aux fonctionnalités du robot. Dans le cas contraire, il en sera aussi averti et il devra re-saisir les champs.

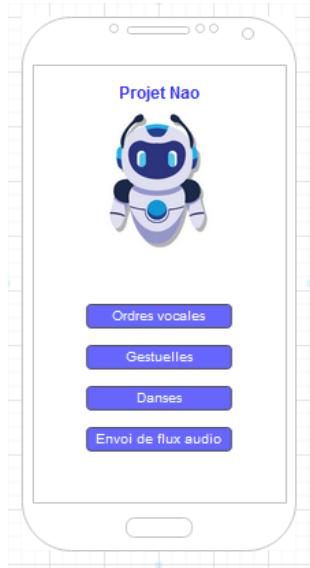
Depuis la page d'accueil, il pourra se déconnecter du robot en appuyant sur le bouton rouge. Une page de confirmation sera affichée pour que l'utilisateur confirme qu'il veut se déconnecter. Il sera possible de revenir sur une page antérieure en utilisant le bouton retour sur les téléphones Android. Il pourra aussi choisir entre deux modes d'utilisations : le mode contrôle et le mode autonome.

Le mode contrôle



Le mode contrôle permet à l'utilisateur de contrôler le robot en utilisant le téléphone comme un casque VR. L'utilisateur aura accès à la vision du robot et pourra bouger sa tête ou se déplacer afin que le robot en fasse de même.

Le mode autonome



Le mode autonome permet au robot d'agir tout seul ou avec des interactions avec l'utilisateur proche du robot. Ce mode sera celui par défaut du robot quand l'utilisateur ne se connecte pas avec le téléphone.

Avec le téléphone, il sera possible de donner des ordres à distance en les donnant depuis le téléphone à l'oral. Il pourra aussi faire des gestes de salutation et autres, des danses. L'utilisateur pourra envoyer des flux audios comme des discours ou même des musiques que le robot jouera.

Livrables

Voici tous les livrables qui devront être rendus à la fin du projet :

- Un rapport de mi-parcours
- Une soutenance de mi-parcours
- L'application Android
- Le programme du robot
- Une documentation
- Un rapport final
- Une soutenance finale
- Une vidéo de présentation du projet

Organisation du groupe

Résumé :

Rôle	Acteur
Chef de projet	Gabriel
Adjoint	Loïck
Organisateur outils	Remy
Médiateur	Lazar
Communicateur	Marius
Monteurs	Haolong, Tianchen
Développeurs	TOUT LE MONDE

Détails :

Gabriel (Chef de projet)

Gabriel occupe le poste de chef de projet, et son rôle principal est de superviser la planification globale du projet. Il est chargé de définir les objectifs, les délais et les priorités du projet. En outre, il assure la coordination de l'équipe en s'assurant que chaque membre sait ce qu'il doit faire et quand le faire. Gabriel est également responsable de la supervision du calendrier du projet, en veillant à ce que toutes les étapes soient respectées en temps et en heure. Son leadership est essentiel pour garantir que le projet avance de manière efficace et organisée.

Loïck (Adjoint)

Loïck occupe le poste d'adjoint, et il soutient activement le chef de projet dans la gestion quotidienne du projet. En cas d'absence du chef de projet, Loïck est prêt à prendre en charge ses responsabilités. Sa collaboration avec les autres membres de l'équipe est cruciale pour garantir la cohésion de l'équipe et la réalisation des objectifs. En tant qu'adjoint, Loïck joue un rôle polyvalent et est prêt à intervenir partout où il est nécessaire pour assurer le succès du projet.

Remy (Organisateur d'outils)

Remy est l'organisateur d'outils, et sa principale responsabilité consiste à gérer les outils, les ressources et les besoins matériels du projet. Il veille à ce que l'équipe dispose des outils nécessaires pour accomplir ses tâches de manière efficace. Remy assure également le suivi des outils de gestion de projet, s'assurant qu'ils sont mis à jour et utilisés de manière optimale. Sa contribution est essentielle pour maintenir les opérations du projet bien organisées.

Lazar (Médiateur)

Lazar joue le rôle de médiateur au sein de l'équipe. Sa principale responsabilité est de faciliter la communication au sein de l'équipe. Il intervient en cas de conflits ou de problèmes interpersonnels et travaille à leur résolution. Lazar crée un environnement de travail harmonieux en encourageant la compréhension mutuelle et en favorisant la résolution de problèmes de manière constructive.

Marius (Communicateur)

Marius assume la position de communicateur, et son rôle consiste à gérer les communications internes et externes du projet. Il est responsable de la diffusion d'informations cruciales aux parties prenantes, en veillant à ce que toutes les informations pertinentes soient transmises de manière efficace et à temps. Marius joue un rôle clé dans la gestion de l'image du projet et de sa visibilité.

Haolong et Tianchen (Monteurs)

Haolong et Tianchen sont les monteurs de l'équipe, et leur principale responsabilité est de capturer des vidéos et des photos du projet à différentes étapes. Ils veillent à ce que toutes les séquences visuelles soient bien organisées, archivées et prêtes à être utilisées pour la documentation du projet. Leur contribution est essentielle pour documenter le progrès du projet de manière visuelle et engageante.

Il est important de noter que, en plus de leurs rôles spécifiques, chaque membre de l'équipe collabore activement et contribue de manière significative à différents aspects du projet. En tant que développeurs informatiques et testeurs, ils apportent des compétences techniques essentielles à la réalisation du projet. Cette polyvalence permet à l'équipe de résoudre des problèmes informatiques, d'optimiser les processus et de garantir la qualité du projet en effectuant des tests et des vérifications approfondis. L'engagement de chaque membre de l'équipe dans ces rôles multiples renforce la capacité de l'équipe à relever les défis du projet de manière proactive et efficace.

Outils d'organisation

Nous allons utiliser plusieurs outils d'organisation qui nous permettront de travailler sur le projet:

- Outils de communication :
 - Discord : il va nous permettre de pouvoir communiquer entre tous les membres de l'équipe et ainsi faire des vocalos si on se retrouve à distance.
 - Google agenda : il nous permettra de planifier les dates importantes, le communicateur peut mettre des dates pour tous les membres de l'équipe.
- Outils de Documentation :
 - La documentation du site internet d'Aldebaran (concepteur du NAO) : C'est avec cette documentation qu'on va se renseigner pour apprendre à utiliser le framework NaoQi.
 - Des tutoriels sur YouTube : Sur cette plateforme, il existe plein de tutoriels pour apprendre à programmer le robot.
 - La documentation de Python : Comme le robot Nao est en python, il nous sera important d'apprendre ce langage.
 - La documentation d'Android Studio : On doit utiliser un téléphone mobile pour pouvoir contrôler le robot à distance. On se référera alors à cette documentation.
 - Forum de communauté : Ces forums, comme Reddit ou StackOverflow, vont nous apporter plein de réponses à nos questions car il y a une très grande communauté de programmeurs avec toute sorte de langage.
 - ChatGPT : cet outil d'IA va nous permettre seulement de nous apporter des réponses si on ne les trouve pas sur internet.
 - Outils de suivie :
 - Gantt Project :
 - Pert : Il nous permettra de calculer nos chemins critiques et retards afin d'optimiser au mieux notre projet.
 - Gantt : Comme on a prévu en avance les tâches qu'on devra faire et le temps qu'il faudra. Alors le Gantt va nous permettre de nous donner une idée du temps qu'il faudra pour effectuer les tâches.
 - Google Drive :
 - Google Sheet : cet outil est très utile pour la gestion des tâches d'un projet car c'est un outil collaboratif en ligne.
 - Google Docs : cet outil permettra de rédiger des documents pour le projet de façon collaborative.
 - GitLab : ce logiciel va nous permettre de stocker tout notre code de façon qu'il soit en sûreté et qu'on puisse revenir à des anciennes sources de codes si on se retrouve bloqués.

Il a été décidé par l'intégralité du groupe en raison de la complexité faible du projet que l'outil de suivi des tâches sera Google Sheet au lieu de Gitlab. En effet, tous les

membres ne sont pas à l'aise avec GitLab. C'est pourquoi, nous avons décidé d'utiliser un tableau plutôt que l'outil de gestion de projet de GitLab.

Tous ces outils seront importants dans notre quotidien pour travailler sur le projet de façon organisée et de façon à ce qu'on avance efficacement.

Liste des tâches techniques

Voici ci-dessous la liste des tâches techniques que nous avons identifiées. Cette liste des tâches est provisoire et subira très certainement de léger ajustement afin de mieux correspondre aux besoins du projet.

Pour réaliser cette liste des tâches, chaque membre du groupe s'est vu attribuer une partie du projet, et devait faire des recherches afin de déterminer les tâches à réaliser. Nous avons ensuite mis en commun les recherches de chacun afin de retirer les doublons (par exemple beaucoup de personnes avaient comme première tâche de se connecter au robot) et établir des dépendances entre chaque partie du projet.

Nom

A - Connexion réseau

- A1 - Installer les bibliothèques/sdk/etc.
- A2 - Connexion au robot

F - Géolocalisation

- F1 - Création d'une application android qui va capturer les données d'accéléromètre + vecteur de mouvement
- F2 - Connecter Nao à cette application pour recevoir les données envoyées

G - Gyroscope

- G1 - Création d'une application android qui va capturer les données des capteurs gyroscopiques et les envoyer vers Nao
- G2 - Connecter Nao à cette application pour recevoir les données envoyées

B - Gestion flux téléphone

- B1 - Envoi d'information du téléphone
- B2 - Récupération d'information du robot

C - Réception et envoi audio à partir de Nao

- C1 - Capture Audio du robot
- C2 - Encodage Audio
- C3 - Envoi audio vers le téléphone
- C4 - Réception audio du téléphone
- C5 - Décodage audio du téléphone
- C6 - Lecture audio du téléphone

D - Réception et envoi audio à partir du Téléphone

- D1 - Capture audio du téléphone
- D2 - Encodage audio
- D3 - Envoi audio vers le robot Nao
- D4 - Réception audio du robot Nao
- D5 - Décodage audio du robot Nao
- D6 - Lecture audio du robot Nao

N - Reconnaissance vocale

- N1 - Installation des prérequis
- N2 - Configuration de l'environnement de développement
- N3 - Installation du module de reconnaissance vocale
- N4 - Création d'une grammaire de reconnaissance vocale

O - Traitement ordres

- O1 - Définition des commandes vocales
- O2 - Création d'une logique de traitement

Nom

- O3 - Programmation des actions
- O4 - Gestion des erreurs et des exceptions
- O5 - Tests et ajustements

K - Mouvement de tête

- K1 - Bouger la tête à droite et à gauche
- K2 - Bouger la tête de haut en bas
- K3 - Interprétation des coordonnées reçus et transformation en donnée utilisable par Nao
- K4 - Mouvement "en temps réel" par rapport au donnée reçus
- K5 - Fluidification du mouvement de la tête
- K6 - Mouvement donnée par la reconnaissance vocale

L - Mouvement jambe bras

- L1 - Bouger le bras à gauche
- L2 - Bouger le bras à droite
- L3 - Bouger la jambe à gauche
- L4 - Bouger la jambe à droite
- L5 - Bouge torse
- L6 - Marcher
- L7 - Se lever
- L8 - Tourner

M - Gestion déplacement jambe bras

- M1 - Interprétation des coordonnées reçus et transformation en donnée utilisable par Nao
- M2 - Transformation des données en mouvement
- M3 - Fluidification des mouvements

P - TTS

- P1 - Sélectionner un moteur de synthèse vocale
- P2 - Installation du moteur TTS
- P3 - Création de fichiers de voix personnalisés
- P4 - Configuration des paramètres TTS
- P5 - Intégration du TTS dans les applications
- P6 - Tests et ajustements

E - Gestion du flux vidéo en direct entre le téléphone et le robot Nao

- E1 - Configuration du flux video depuis Nao
- E2 - Mise en place d'un serveur vidéo sur le robot Nao
- E3 - Développement de l'application Android pour la réception du flux vidéo
- E4 - Test et intégration du flux vidéo en direct

Nom

Q - Reconnaissance objet

- Q1 - Configuration de la reconnaissance d'objets ronds
- Q2 - Développement de l'algorithme dans le robot Nao
- Q3 - Intégration de l'algorithme dans le robot Nao
- Q4 - Test de la reconnaissance d'objets ronds

H - Livrable - Maquette

- H1 - Maquette de la page connexion
- H2 - Maquette de la page menu
- H3 - Maquette de la page utilisation
- H4 - Validation des maquettes

I - Livrable - Developpement interface

- I1 - Développement de la page connexion
- I2 - Développement de la page menu
- I3 - Développement de la page utilisation
- I4 - Développement de la navigation entre les pages
- I5 - Valider les interfaces

J - Livrable - Gestion des paramètres

- J1 - Ajout et test de la fonctionnalité de connexion au robot
- J3 - Ajout des fonctionnalités de la page utilisation
- J2 - Ajout et test des fonctionnalités du menu
- J4 - Test intégral

Description des tâches techniques

Nous allons ci-dessous décrire chaque tâche technique en détail et comment nous comptons les aborder.

Liste des tâches

1. Téléphone

1. 1. Connexion réseau

Pour permettre la connection entre le code java sur Android-Studio et le code sur le robot NAO, nous allons devoir utiliser différentes bibliothèques comme “socket”, “multithreading” tout en ajoutant les droits de connections à Android-Studio, aux ordinateurs et au NAO. Ensuite un système de serveur-client solide devra être mis en place afin de permettre un bon échange pour les diverses données qui circuleront entre le NAO et le téléphone comme par exemple le flux vidéo ou le flux audio. Pour alléger le nombre d'opérations sur le robot NAO, nous allons programmer les serveurs sur le téléphone et les clients sur le robot, cela permet de ne pas surcharger la mémoire RAM du robot.

1. 2. Gestion de flux (téléphone)

L'envoi d'informations depuis un téléphone vers un robot, ainsi que la récupération d'informations du robot vers le téléphone, dépendent de la manière dont nous allons configurer la communication entre les deux appareils. Pour permettre la communication entre le téléphone et le robot, les deux appareils doivent être connectés au même réseau. Cela peut être un réseau local (Wi-Fi) ou, dans certains cas, une connexion via Internet. Nous devons définir un protocole de communication entre le téléphone et le robot. Pour envoyer des informations depuis un téléphone vers le robot, il faudra créer un script via Android Studio qui permettra de créer les données à envoyer, par exemple des commandes ou des données sensorielles et puis de les envoyer au robot via le protocole de communication que nous aurons défini. Cela peut être réalisé en utilisant des bibliothèques de communication, telles que des API REST pour HTTP ou en utilisant des connexions socket personnalisées. Pour récupérer des informations du robot vers le téléphone, le robot doit être configuré pour envoyer ces informations au téléphone. Nous devrons coder en python (via NAOqi) un script qui permettra d'exposer les données que nous souhaitons récupérer et les envoyer au téléphone via le protocole de communication que l'on aura défini. Enfin, nous intégrerons le script de communication dans notre application mobile et robotique respective. Cela peut inclure la création d'une interface utilisateur sur le téléphone pour interagir avec le robot et l'interprétation des données reçues du robot.

1. 3. Réception et envoi audio

La capture audio, l'encodage, l'envoi, la réception, le décodage et la lecture audio impliquent une communication audio bidirectionnelle entre le robot NAO et un téléphone. Pour la capture audio, le robot NAO est équipé de microphones et le logiciel NAOqi offre des API pour accéder aux données audios capturées par les microphones. Nous pourrons utiliser ces API pour enregistrer l'audio ambiant. Les données audios capturées sur le robot doivent être encodées dans un format approprié pour la transmission. Les formats courants pour l'encodage audio incluent WAV, MP3, etc. Nous pouvons utiliser des bibliothèques de traitement audio en Java pour effectuer l'encodage. Pour l'envoi audio depuis le robot vers le téléphone nous devrons établir une connexion réseau entre le robot NAO et le téléphone, comme expliqué précédemment. Nous pourrons utiliser le protocole de communication que nous aurons défini pour transmettre les données audios encodées depuis le robot vers le téléphone. Cela peut se faire via une connexion TCP/IP, WebSocket, ou un autre protocole personnalisé. Le téléphone doit être capable de recevoir les données audios transmises par le robot via le même protocole de communication. Nous devrons coder un script sur le téléphone afin de recevoir les données audios, les stocker temporairement et les préparer pour le décodage. Les données audios reçues sur le téléphone doivent être décodées dans un format lisible par le téléphone, par exemple WAV ou MP3. Il faudra utiliser des bibliothèques de décodage audio sur le téléphone pour effectuer cette opération. Les bibliothèques telles que Android MediaCodec peuvent être utiles sur la plateforme Android. Une fois les données audios décodées, nous pouvons les lire à l'aide des fonctionnalités audio natives du téléphone. Sur Android, par exemple, nous pourrons utiliser les API audio Android pour la lecture. Il est important de noter que la latence et la qualité audio sont des facteurs critiques dans ce processus. Nous devrons prendre en compte des problèmes tels que la compression audio, la synchronisation audio, la gestion des erreurs de transmission et la gestion de la qualité audio pour une expérience utilisateur optimale.

1.4. Gestion de flux vidéo en direct entre le téléphone et Nao

La configuration d'un flux vidéo depuis le robot NAO vers un appareil Android nécessite plusieurs étapes, notamment la configuration du serveur vidéo sur le robot NAO, le développement d'une application Android pour la réception du flux vidéo en direct, et des tests d'intégration. Sur le robot NAO, nous pourrons utiliser les caméras intégrées pour capturer le flux vidéo en direct et le logiciel NAOqi propose des API pour accéder au flux vidéo des caméras. Nous pourrons utiliser ces API pour configurer la capture vidéo. Nous devrons ensuite créer un serveur vidéo sur le robot NAO qui diffuse le flux vidéo capturé en direct et utiliser les bibliothèques appropriées pour créer ce serveur vidéo. Par exemple, nous pourrons utiliser des bibliothèques comme OpenCV ou GStreamer pour la manipulation de la vidéo sur le robot NAO. Nous développerons une application Android qui peut recevoir et afficher le flux vidéo en direct du robot NAO. Nous devrons configurer la connexion réseau entre l'application Android et le serveur vidéo sur le robot. Les protocoles courants pour le streaming vidéo incluent RTSP ou HTTP. Nous devrons nous assurer que la connexion réseau entre le robot NAO et l'application Android est établie correctement puis tester le streaming vidéo pour nous assurer que la qualité de la vidéo est suffisamment bonne et que la latence est acceptable. Nous intégrerons le flux vidéo dans l'interface

utilisateur de notre application Android de manière à ce qu'il puisse être affiché en temps réel sur l'écran du téléphone. Nous nous assurerons de gérer les problèmes liés à la qualité de la vidéo, à la compression vidéo, le cas échéant, et à la résolution pour que l'expérience utilisateur soit fluide. L'intégration du flux vidéo en direct peut être complexe, en particulier si nous souhaitons transmettre un flux de haute qualité avec une latence minimale. Nous devrons peut-être également prendre en compte des problèmes de bande passante réseau, de compatibilité de codec vidéo et de gestion des erreurs de transmission.

1.5. Géolocalisation

Pour créer une application Android capable de capturer les données de l'accéléromètre et des vecteurs de mouvement, et de connecter le robot NAO à cette application pour recevoir les données envoyées, nous installerons Android Studio et s'assurer que notre robot NAO est configuré et connecté au même réseau que l'appareil Android. Il faudra ensuite créer un projet Android dans Android Studio, intégrer le script pour capturer les données de l'accéléromètre et des vecteurs de mouvement à l'intérieur de notre application Android. Nous pourrons utiliser le capteur d'accéléromètre intégré à l'appareil Android pour cela et créer une interface utilisateur pour afficher ou transmettre ces données à Nao. Ensuite une connexion réseau entre l'application Android et le robot NAO sera établie. Nous pourrons utiliser des sockets, des protocoles de communication comme HTTP ou WebSocket, ou des bibliothèques tierces pour la communication. Une fois les données d'accéléromètre et de vecteurs de mouvement capturées, ils seront encodés pour l'envoi au robot NAO. Il peut s'agir de données sous forme de texte ou de données structurées, par exemple .JSON. Un script sera codé sur le robot NAO pour recevoir les données envoyées par l'application Android. Le robot NAO devra également être configuré pour interpréter ces données. Une fois que les données sont reçues par le robot NAO, nous pourrons les traiter en conséquence. Par exemple, nous pourrons utiliser les données de l'accéléromètre pour ajuster le comportement du robot ou pour déclencher des actions spécifiques. Nous intégrerons les données d'accéléromètre et de vecteurs de mouvement dans le comportement du robot NAO. Nous devrons utiliser les API NAOqi pour contrôler les mouvements et les actions du robot en fonction de ces données. Nous testerons notre application Android et le robot NAO pour nous assurer que la communication fonctionne correctement. Nous effectuerons des tests de performance pour nous assurer que la latence est acceptable et que les données sont transmises de manière fiable.

1.6. Gyroscope

Pour créer une application Android qui capture les données des capteurs gyroscopiques de l'appareil et envoie ces données au robot NAO, il faut installer Android Studio et s'assurer que notre robot NAO est configuré et connecté au même réseau que l'appareil Android. Il faut créer un projet Android dans Android Studio puis intégrer le code pour capturer les données des capteurs gyroscopiques, par exemple l'accéléromètre à l'intérieur de notre application Android en utilisant les API fournies par Android. Les capteurs gyroscopiques peuvent fournir des informations sur l'orientation et les mouvements de

l'appareil. Nous devrons établir une connexion réseau entre l'application Android et le robot NAO. Nous pourrons utiliser des sockets, des protocoles de communication comme HTTP ou WebSocket, ou des bibliothèques tierces pour la communication. Une fois les données gyroscopiques capturées, nous nous préparerons pour l'envoi au robot NAO. Il peut s'agir de données sous forme de texte ou de données structurées, par exemple .JSON. Nous nous assurerons de formater les données de manière à ce que le robot NAO puisse les comprendre. Un script sera codé sur le robot NAO pour recevoir les données envoyées par l'application Android. Le robot NAO devra également être configuré pour interpréter ces données. Une fois que les données sont reçues par le robot NAO nous pourrons les traiter en conséquence. Par exemple, nous pouvons utiliser les données gyroscopiques pour ajuster le comportement du robot ou pour déclencher des actions spécifiques. Nous devrons intégrer les données gyroscopiques dans le comportement du robot NAO en utilisant les API NAOqi pour contrôler les mouvements et les actions du robot en fonction de ces données. Nous testerons notre application Android et le script sur le robot NAO pour nous assurer que la communication fonctionne correctement et effectuer des tests de performance pour nous assurer que la latence est acceptable et que les données sont transmises de manière fiable.

1. 7. Livrable - Maquette

La création de maquettes pour les pages de connexion, de menu et d'utilisation (VR) dans une application implique de concevoir visuellement l'interface utilisateur de chaque écran. La maquette de la page de connexion est une représentation visuelle de l'écran de connexion de notre application. Elle comprend généralement des éléments tels qu'un champ de saisie pour le nom d'utilisateur, un champ de saisie pour le mot de passe, des boutons de connexion et d'inscription, et peut-être des liens pour la récupération de mot de passe ou pour changer de langue. La conception de la maquette doit prendre en compte l'expérience utilisateur, en s'assurant que les éléments sont bien disposés et faciles à utiliser. Les couleurs, les polices, et les images doivent être cohérentes avec le style de votre application. La maquette doit également tenir compte des spécificités de la plateforme sur laquelle l'application sera exécutée (par exemple, Android ou iOS). La maquette de la page du menu représente l'écran principal de notre application où les utilisateurs peuvent accéder à diverses fonctionnalités ou sections. Cela peut inclure un menu latéral, un menu en bas d'écran, ou d'autres éléments de navigation. Nous devrons organiser les éléments du menu de manière à ce qu'ils soient faciles à parcourir et à sélectionner. Les icônes et les libellés doivent être clairs pour que les utilisateurs puissent identifier rapidement les sections ou les fonctionnalités de l'application. La maquette de la page d'utilisation en VR représente l'écran VR de l'application, où les utilisateurs interagissent avec un environnement en 3D. Cette maquette peut inclure des captures d'écran de l'environnement VR, des modèles 3D, des icônes et des contrôles d'interaction. Nous nous assurons de mettre en évidence les éléments clés de l'expérience VR, tels que les contrôles de mouvement, les instructions, les repères visuels, et tout ce qui est nécessaire pour que l'utilisateur se familiarise avec l'interaction en VR.

1.8. Livrable – Développement interface

Avant de commencer à développer l'application, il est essentiel de valider les maquettes. Cela implique généralement de les montrer à l'équipe de développement, aux parties prenantes et, si possible, à des utilisateurs potentiels. Recueillir des commentaires sur la conception des maquettes pour nous assurer que l'interface utilisateur est intuitive, attrayante et qu'elle répond aux besoins de l'application. Lorsque les maquettes sont validées, elles serviront de base solide pour le développement de l'application. Toute modification ultérieure de la conception peut être plus coûteuse, il est donc préférable de s'assurer que les maquettes sont satisfaisantes dès le départ. Le développement d'une application, en particulier pour un casque de réalité virtuelle (VR), implique la création de plusieurs pages, la navigation entre elles, et la validation des interfaces. Le développement de la page de connexion consiste à créer l'écran de connexion de l'application. Nous devrons concevoir et intégrer les éléments de l'interface utilisateur, tels que les champs de saisie pour le nom d'utilisateur et le mot de passe, les boutons de connexion et d'inscription, et les éléments d'interaction (comme le pointage dans l'espace VR). Nous programmerons également la logique de connexion, qui peut inclure la vérification des informations d'identification, l'interaction avec un serveur d'authentification, et la gestion des sessions utilisateur. La page du menu est généralement l'écran principal de l'application VR, où nous présenterons les options de navigation et les fonctionnalités de l'application. Nous développerons cette page en créant des éléments de menu interactifs en 3D qui réagissent aux mouvements et aux gestes de l'utilisateur. Nous utiliserons des bibliothèques de développement VR pour gérer ces interactions. Les éléments du menu doivent être conçus pour être intuitifs et faciles à utiliser dans un environnement VR. La page d'utilisation en VR est l'endroit où les utilisateurs interagissent avec l'environnement en 3D de l'application. Nous développerons cette page en créant des modèles 3D, des textures, des animations, et en programmant les interactions de l'utilisateur avec l'environnement virtuel. Nous nous assurons de fournir des instructions claires et des repères visuels pour guider les utilisateurs dans leur expérience VR. La navigation entre les pages en VR peut être différente de la navigation sur un écran traditionnel. Elle peut se faire via des interactions gestuelles, des boutons virtuels, ou en regardant et pointant vers des éléments interactifs. Nous développerons la navigation entre les pages en programmant des transitions fluides et en utilisant des éléments de navigation appropriés pour un environnement VR. La validation des interfaces VR est essentielle pour garantir que l'expérience utilisateur est fluide et intuitive. Nous testerons notre application VR sur des casques réels pour nous assurer que les interactions et les performances sont conformes à nos attentes. Nous nous assurerons que les éléments de menu et les transitions entre les pages fonctionnent correctement et que l'application est exempte de bugs ou de problèmes de performance.

1.9. Livrable – Gestion des paramètres

L'ajout et le test de fonctionnalités pour une application qui se connecte à un robot, comprend un menu, et offre une expérience en casque VR nécessitant plusieurs étapes, y compris le développement, le test de chaque fonctionnalité individuelle, puis le test intégral de l'ensemble de l'application. Il faut intégrer la fonctionnalité de connexion au robot dans notre application. Cela inclut l'établissement d'une connexion réseau avec le robot, la saisie

des informations d'identification, par exemple nom d'utilisateur et mot de passe, la gestion des erreurs de connexion, etc. Nous testerons la fonctionnalité de connexion de manière isolée pour nous assurer qu'elle fonctionne correctement. Nous vérifierons que la connexion au robot est établie et que les informations d'identification sont vérifiées avec succès. Nous gérerons les scénarios d'erreur, tels que les échecs de connexion. Il faudra intégrer les fonctionnalités du menu, telles que la navigation entre différentes sections de l'application, la sélection d'options, la personnalisation du menu, etc. et tester chaque fonctionnalité du menu individuellement. Nous nous assurerons que les éléments de menu réagissent correctement aux interactions de l'utilisateur, que la navigation fonctionne de manière fluide et que les informations sont correctement affichées. Nous développerons les fonctionnalités pour l'expérience en casque VR. Cela peut inclure la création d'environnements 3D, l'intégration de modèles 3D, la programmation des interactions, les animations, etc. Nous testerons ces fonctionnalités en VR et s'assureront que l'environnement VR est bien rendu et que les interactions se comportent comme prévu. Nous vérifierons que l'expérience en VR est immersive et répond aux attentes. Une fois qu'on aura testé individuellement chaque fonctionnalité (connexion au robot, menu, expérience en VR), nous effectuerons un test intégral de l'ensemble de l'application afin de s'assurer que toutes les fonctionnalités fonctionnent ensemble harmonieusement. Nous testerons la navigation depuis la page de connexion vers le menu et de là vers l'expérience en VR. Nous identifierons et corrigerais tout problème d'intégration ou d'interaction entre les fonctionnalités. Nous validerons l'ensemble de l'interface utilisateur en nous assurant que les éléments de menu et d'interaction en VR sont cohérents et intuitifs. Nous observerons l'expérience utilisateur pour identifier des points faibles et apporter des améliorations si nécessaire. Nous effectuerons des tests de performances pour nous assurer que l'application fonctionne de manière fluide et qu'elle est réactive.

2. Nao

2.1. Mouvement tête

Le mouvement de la tête du robot NAO peut être contrôlé par des moteurs intégrés. Pour faire bouger la tête à droite, à gauche, de haut en bas, ou pour interpréter les coordonnées reçues et les transformer en données utilisables par NAO, les moteurs de la tête de NAO peuvent être contrôlés en utilisant les API NAOqi fournies par SoftBank Robotics. Il est possible définir l'angle de chaque moteur pour orienter la tête dans la direction souhaitée. Par exemple, pour faire tourner la tête à droite ou à gauche, nous pourrons utiliser une commande comme `motionProxy.setAngles("HeadYaw", angle)` où `angle` est la valeur en radians. Pour faire bouger la tête de haut en bas, nous utiliserons une commande similaire en ajustant les angles appropriés. Pour interpréter les coordonnées reçues, par exemple des coordonnées x, y, nous devrions avoir une logique dans notre application qui traduit ces coordonnées en mouvements de la tête. Cela peut inclure la conversion des coordonnées en angles appropriés pour les moteurs de la tête, par exemple, si nous recevons des coordonnées x, y pour la direction de la tête, nous pouvons utiliser des formules mathématiques pour calculer les angles des moteurs. Nous pourrons configurer un mécanisme qui ajuste constamment les angles des moteurs de la tête en fonction des nouvelles données reçues. Cela permettrait de suivre les coordonnées en "temps réel". Nous pourrons utiliser des boucles de contrôle ou des événements pour surveiller les nouvelles données et ajuster en conséquence. Pour obtenir un mouvement fluide de la tête, nous pouvons définir des transitions douces entre les angles des moteurs. Par exemple, au lieu de définir instantanément de nouveaux angles, nous pouvons créer des transitions avec des déplacements progressifs. Nous utiliserons des bibliothèques de mouvement fluide ou des fonctions de déplacement interpolées pour atteindre cette fluidité. Si le mouvement de la tête est lié à la reconnaissance vocale, nous pourrons configurer notre application pour que le robot NAO tourne ou incline la tête en fonction des commandes vocales reconnues. L'intégration de la reconnaissance vocale nécessite généralement l'utilisation d'une bibliothèque ou d'un service tiers pour traiter les commandes vocales.

2.2. Mouvement jambes / bras

Le mouvement du robot NAO, y compris le mouvement des bras, des jambes, du torse, la marche, le fait de se lever, et la rotation, est contrôlé par les moteurs internes du robot et est généralement géré à l'aide de l'API NAOqi fournie par SoftBank Robotics. Pour déplacer les bras de NAO, nous utilisons les API NAOqi pour contrôler les moteurs de bras. Nous pouvons spécifier l'angle de chaque articulation du bras (épaule, coude, poignet) pour définir la position souhaitée du bras. Par exemple, pour lever le bras gauche, nous pouvons utiliser `motionProxy.setAngles("LShoulderPitch", angle)` où "LShoulderPitch" est le nom de l'articulation de l'épaule gauche, et `angle` est l'angle souhaité en radians. Les jambes de NAO peuvent également être contrôlées en utilisant les API NAOqi. Nous spécifions les angles des articulations des jambes (hanche, genou, cheville) pour déterminer la position et l'orientation des jambes. Par exemple, pour plier la jambe droite, nous pouvons utiliser `motionProxy.setAngles("RHipPitch", angle)` où "RHipPitch" est l'articulation de la hanche droite. Le torse de NAO peut être incliné ou tourné en utilisant les API appropriées. Nous

spécifions les angles des articulations du torse pour contrôler son orientation. Par exemple, pour incliner le torse, nous pouvons utiliser `motionProxy.setAngles("TorsoPitch", angle)`. Pour faire marcher NAO, nous utilisons une combinaison de mouvements coordonnés des jambes, des bras, et du torse. Nous pouvons spécifier la séquence de pas, la direction, la vitesse de la marche, et d'autres paramètres pour définir le mouvement de marche souhaité. Il existe des bibliothèques et des fonctions prédéfinies pour simplifier le contrôle de la marche. NAO peut se lever depuis une position assise en utilisant les moteurs des jambes et des bras. Nous devrons définir une séquence de mouvements qui coordonne l'extension des jambes et le mouvement des bras pour permettre au robot de se lever de manière stable. Nous pourrons également utiliser des fonctions prédéfinies pour effectuer cette action. Pour faire tourner NAO, nous pouvons ajuster les angles des articulations des jambes pour changer sa direction. Nous pouvons également incliner le torse dans la direction souhaitée pour un meilleur équilibre. Le contrôle précis de la rotation dépend de la stratégie de mouvement que nous utiliserons, que ce soit en mode stationnaire ou en mode marche.

2. 3. Gestion déplacement jambes / bras

L'interprétation des coordonnées reçues et leur transformation en données utilisables par le robot NAO, ainsi que la transformation de ces données en mouvement fluide, nécessite une approche méthodique. L'interprétation des coordonnées dépend du contexte de notre application. Si nous recevons des coordonnées de sources externes, par exemple, des capteurs, des commandes utilisateur, des données GPS, nous devons comprendre ce que ces coordonnées représentent. Par exemple, les coordonnées pourraient indiquer une position dans un espace 2D ou 3D, des mouvements relatifs par rapport à une position de départ, ou d'autres informations géospatiales. Pour transformer ces coordonnées en données utilisables par le robot NAO, nous devons les convertir en valeurs compréhensibles par les moteurs du robot. Cela implique généralement de définir des angles ou des paramètres spécifiques que NAO peut interpréter. Par exemple, si les coordonnées indiquent une direction, nous pouvons les convertir en angles d'articulation des moteurs du robot pour que la tête, les bras, ou les jambes bougent dans la direction souhaitée. Une fois que nous avons des données compréhensibles par NAO, nous pouvons utiliser les API NAOqi pour définir les mouvements du robot. Cela peut impliquer de définir les angles des articulations, les vitesses, les accélérations, etc., en fonction des données reçues. Par exemple, si nous avons des données de direction pour un mouvement de la tête, nous utilisons les API NAOqi pour définir les angles des moteurs de la tête pour orienter la tête dans cette direction. La fluidification des mouvements est importante pour éviter que le robot ne bouge brusquement et de manière saccadée. Pour obtenir des mouvements fluides, nous pouvons définir des transitions douces entre les positions des moteurs. Nous utiliserons des bibliothèques de mouvement fluide ou des techniques d'interpolation pour créer des trajectoires de mouvement en douceur, en particulier lorsque le robot doit effectuer des mouvements complexes ou en temps réel. Une étape essentielle consiste à tester les mouvements sur le robot NAO pour nous assurer qu'ils sont fluides et précis. Nous pouvons affiner les paramètres, les courbes de mouvement et les transitions en fonction des performances réelles du robot. Il est important de prendre en compte les limites physiques du robot pour éviter des mouvements inappropriés.

3. Interprétation

3. 1. Reconnaissance vocale

Plusieurs étapes sont essentielles pour intégrer la reconnaissance vocale dans notre application. Nous nous assurerons d'avoir un environnement de développement configuré pour travailler avec le robot NAO, ce qui nécessite l'installation d'outils de développement tels que NAOqi SDK, Python, et d'autres dépendances spécifiques au robot. Nous configurerons notre environnement de développement en utilisant les outils et les bibliothèques fournis par SoftBank Robotics. Cela peut inclure l'installation du logiciel NAOqi SDK, la configuration de l'accès au robot via le réseau, et la mise en place d'un environnement de programmation Python pour interagir avec le robot. SoftBank Robotics fournit un module de reconnaissance vocale appelé "ALSpeechRecognition" qui peut être installé sur le robot. Nous devrons intégrer ce module dans notre environnement de développement. Le module de reconnaissance vocale permet d'enregistrer la parole, de la transcrire en texte, et d'interpréter les commandes vocales. Une fois que le module de reconnaissance vocale installé, nous pourrons créer une grammaire qui définit les commandes vocales que le robot doit reconnaître. La grammaire est généralement définie sous forme de règles, qui spécifient les mots ou les phrases attendus. Nous définirons des callbacks pour gérer les événements liés à la reconnaissance vocale, tels que la détection de commandes vocales. Lorsqu'une commande vocale est reconnue, nous pourrons déclencher des actions spécifiques sur le robot, telles que le mouvement de la tête, la marche, la réponse vocale, etc. Nous testerons la reconnaissance vocale en utilisant notre grammaire et nous nous assurerons que le robot reconnaît correctement les commandes vocales.

3. 2. Traitement des ordres

La mise en place de la reconnaissance vocale implique la définition des commandes vocales, la création d'une logique de traitement, la programmation des actions en réponse à ces commandes, la gestion des erreurs et des exceptions, ainsi que des tests et ajustements pour garantir au bon fonctionnement du robot NAO. Pour commencer, nous définirons les commandes vocales que nous souhaitons que le robot reconnaîsse. Ces commandes correspondent aux instructions que les utilisateurs prononceront pour interagir avec le robot. Les commandes vocales peuvent être des mots, des phrases ou des expressions spécifiques que le robot doit comprendre. Par exemple, nous pouvons définir des commandes personnalisables telles que "Avance", "Tourne à gauche", "Dis bonjour". Une fois que les commandes vocales sont définies, nous créons une logique de traitement pour gérer ces commandes. Cette logique détermine ce que le robot doit faire en réponse à chaque commande. Nous associerons chaque commande vocale à une action spécifique, comme le mouvement du robot, la parole, l'affichage d'informations à l'écran. Nous programmerons les actions associées à chaque commande vocale en utilisant les API NAOqi. Par exemple, si la commande vocale est "Avance", le robot effectuera l'action avancer. Les actions peuvent inclure le mouvement des membres du robot, la communication verbale, l'affichage d'animations, la réactivité aux commandes et d'autres fonctionnalités. Nous prévoirons des mécanismes de gestion des erreurs et des exceptions

pour traiter les situations où le robot ne comprend pas correctement la commande ou ne peut pas l'exécuter. Par exemple, si le robot ne reconnaît pas une commande, il demandera une clarification ou donnera des instructions alternatives. Nous testerons l'ensemble du système en utilisant des commandes vocales pour nous assurer que le robot comprend les instructions et réagit correctement. Pendant les tests, nous surveillerons les réponses du robot et ajusterons la logique de traitement ou la grammaire des commandes si nécessaire pour améliorer la précision et la convivialité.

3. 3. TTS (Text To Speech)

Le choix et la mise en place d'un moteur de synthèse vocale (TTS, Text-to-Speech) pour le robot NAO impliquent plusieurs étapes, la sélection et l'installation du moteur, la création de fichiers de voix personnalisés, la configuration des paramètres, et les tests. Nous choisirons un moteur de synthèse vocale qui est compatible avec le robot NAO et répond à nos besoins en termes de qualité vocale, de langues prises en charge et de fonctionnalités. SoftBank Robotics fournit généralement des recommandations pour les moteurs TTS compatibles que nous choisirons d'installer sur le robot NAO en suivant les instructions fournies par le fabricant du moteur. Nous nous assurerons que le moteur choisi au préalable est correctement configuré pour être utilisé par le robot. Si nous souhaitons utiliser des voix personnalisées pour le robot NAO, nous devrons créer des fichiers de voix spécifiques comme l'enregistrement de voix humaines ou la génération de voix synthétiques. Les fichiers de voix personnalisés doivent être compatibles avec le moteur. Nous devrons configurer les paramètres TTS en fonction de nos besoins. Cela peut inclure le choix de la langue, le type de voix, la vitesse de lecture sur la parole, de l'intonation, et d'autres paramètres consultables sur la documentation officielle. Nous testerons la synthèse vocale en utilisant des scripts ou des commandes pour nous assurer que tout fonctionne correctement. Nous écouterons la voix générée pour évaluer la qualité et l'intonation. Si nous créons des voix personnalisées, nous nous assurerons qu'elles sont bien intégrées.

3. 4. Reconnaissance d'objet

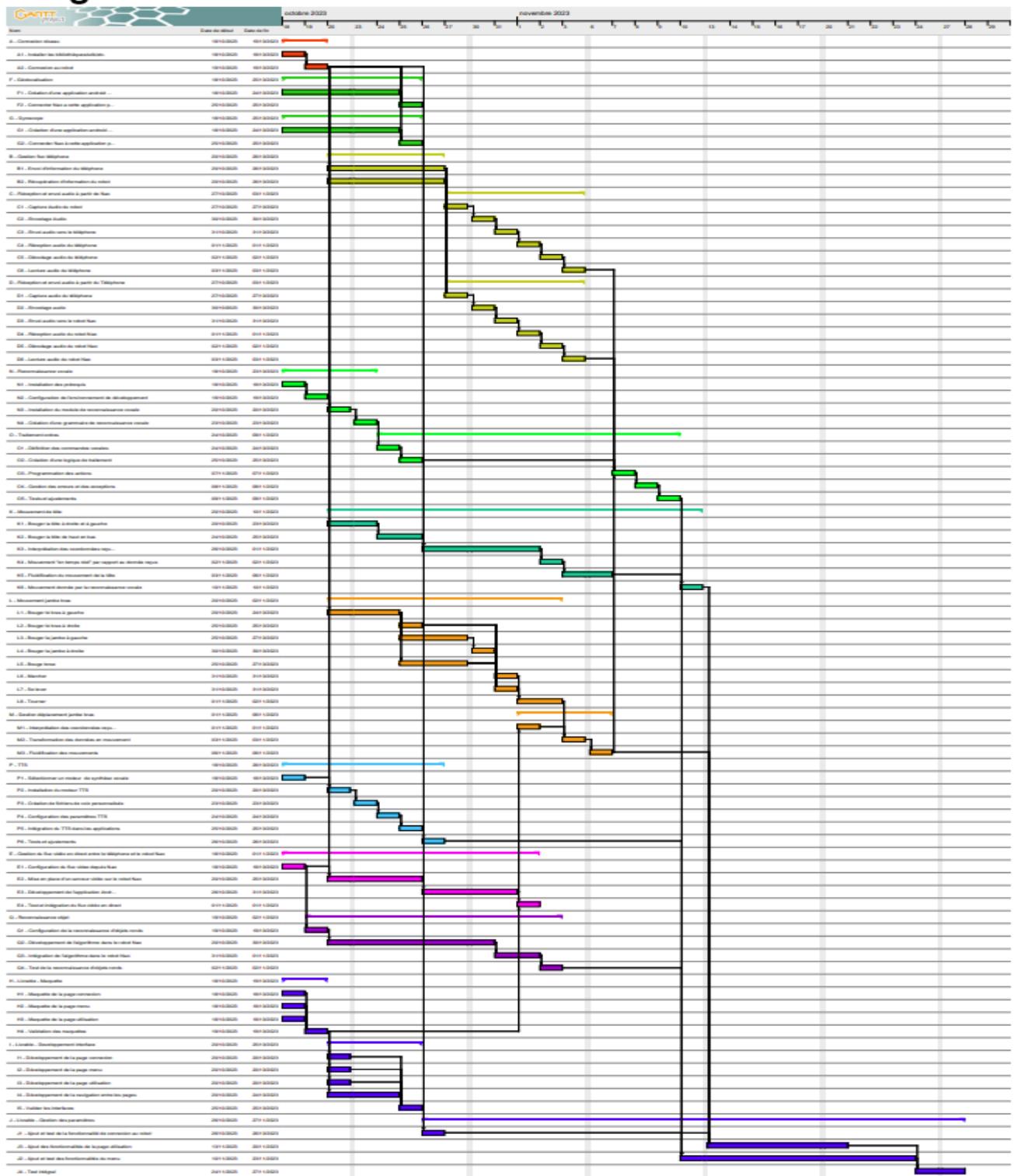
Cette partie est facultative et sera développée uniquement si tout le reste est terminé. La configuration, le développement de l'algorithme, l'intégration de l'algorithme dans le robot NAO, et les tests liés à la reconnaissance d'objets sont des étapes essentielles pour permettre au robot de détecter et de réagir. Avant de commencer, nous configurons le robot NAO pour utiliser des capteurs appropriés tels que des caméras ou des capteurs de profondeur pour la détection. Nous nous assurerons ensuite que les paramètres du robot sont correctement paramétrés pour l'optimisation, tels que la résolution de la caméra, la fréquence d'échantillonnage, et ainsi les autres capteurs.. Nous développerons un algorithme de reconnaissance d'objets en utilisant des techniques de traitement d'image et de vision par ordinateur. Cet algorithme devrait permettre d'identifier les objets dans les données capturées par les capteurs du robot. Les techniques peuvent inclure la détection de cercles, la segmentation d'image, la classification d'objets, etc. Nous intégrerons l'algorithme de reconnaissance dans le logiciel du robot NAO en utilisant les API appropriées, telles que les API NAOqi. Nous créerons des modules logiciels qui permettent au robot de traiter les données des capteurs en temps réel, d'appliquer l'algorithme de reconnaissance, et de

réagir en conséquence. Nous effectuerons des tests sur le robot pour évaluer la capacité de reconnaissance d'objets. Nous présenterons des objets disposés à différentes distances, angles et conditions d'éclairage pour évaluer les performances de l'algorithme. Nous vérifierons que le robot est capable de détecter et de suivre ces objets, ainsi que de réagir en fonction de la reconnaissance, par exemple en déplaçant la tête pour suivre l'objet. Si l'algorithme ne fonctionne pas correctement, nous apporterons des ajustements et des optimisations. Cela peut inclure des changements dans les paramètres de l'algorithme, l'ajout de filtres pour améliorer la robustesse, ou l'amélioration de l'efficacité du traitement d'image. Nous effectuerons des itérations de tests et d'ajustements pour améliorer les performances de la reconnaissance.

GANTT

Voici le diagramme de GANTT résultant de la liste des tâches techniques établie plus tôt. Il a été réalisé à l'aide du logiciel GanttProject. La majeure difficulté était d'organiser le diagramme afin que les dépendances soit lisible, car beaucoup de dépendances se croisent. <https://drive.google.com/file/d/1gq9vHMqkpgF27tBNI5nA1Wz-jspEk7d2/view?usp=sharing>

Diagramme de Gantt



Développement du Nao

Répartition des tâches techniques

Tâche	Responsables
Gestion de la connectivité réseau	Moïra, Lazar, Remy, Marius
Gestion des flux de données (Vidéo et audio)	Moïra, Lazar, Remy, Marius
Réception et envoi de l'audio	Moïra
Géolocalisation et gyroscope	Lazar
Maquettage et développement des interfaces de l'application Android	Remy
Mouvements du robot (Tête, jambes et bras)	Gabriel, Loïck, Léo et Tiancheng
Reconnaissance vocale et TextToSpeech	Keryann, Haolong
Reconnaissance d'objets	Marius

Cette répartition des tâches montre comment chaque membre de l'équipe contribue de manière significative à notre projet global, en combinant ses compétences pour créer un robot interactif et fonctionnel. Chaque domaine de responsabilité est crucial pour atteindre nos objectifs et offrir une expérience utilisateur exceptionnelle.

RÉALISATION	
Récupération + envoie des données gyro	Lazar
Récupération + envoie des données geoloc	Lazar
Récupération des images de la caméra du nao	Loick
Transformation des images en flux vidéo	Loïck, Marius, Gabriel
Envoie du flux vidéo au serveur java	Loïck, Marius
Création de serveurs Java (Eclipse) de réception et affichage des images	Marius
Création de serveurs Java (Eclipse) d'envoie de donnée vers nao	Marius
Récupération + envoie du son que capte le nao	Marius (Gaétan)
Codage de l'interface de l'appli sur Android Studio	Remy
Création et optimisation d'une solution de reconnaissance vocale sur un téléphone	Moira
Optimisation de la solution pour qu'elle écoute en permanence/en boucle	Moira

Envoie des mots reconnus au robot en String	Moira
Ecriture des fonctions permettant de déplacer les jambes	Léo, Tiancheng
Traiter les données du gyro./geoloc. pour déplacement correctement les jambes	Léo
Ecriture des fonctions permettant de déplacer les bras	Tiancheng
Ecriture des fonctions permettant de déplacer la tête	Gabriel
Traiter les données du gyro./geoloc. pour tourner correctement la tête	Gabriel, Loick
Ecriture des fonctions permettant de faire parler le robot (avec un String en entrée)	Keryann
Montage/Réalisation de la vidéo	Haolong

Liste des outils de programmation

Afin de pouvoir réaliser notre projet, le groupe s'était donc réuni dans une réunion, pour savoir quels outils et logiciels, il faudrait utiliser, sachant que la contrainte du projet étant que le projet doit être réalisé en Python pour la partie NAO mais aussi, qu'on doit créer une application Android, afin de pouvoir communiquer avec le NAO de partout. On a donc décidé d'utiliser les outils suivants:

- **Éditeur de code et environnement de développement**

Pour ce qui est des éditeurs de code, à savoir les logiciels pour pouvoir taper et composer nos lignes de code, on utilisera :

- **Visual Studio Code**, un éditeur de code (ou IDE pour environnement de développement intégré) permettant de créer ou modifier du code, cet outil est développé et propulsé par Microsoft. Ce logiciel permet notamment de coder dans plusieurs langages, tels que Python, Java et plein d'autres.
- **Apache**, un autre éditeur de code pour composer du code en Python développé et propulsé par JetBrains, l'avantage de PyCharm, c'est qu'il possède un compilateur intégré ce qui est utile pour tester, afin de voir si le code fonctionne correctement, mais aussi à compiler le code afin de le transmettre au NAO.
- **Android Studio**, créée par Google et JetBrains, il s'agit d'un IDE permettant de pouvoir créer du code Java (ou Kotlin) pour mettre en oeuvre une application Android, et ainsi que la création d'interfaces graphiques pour l'application Android via l'XML, mais aussi, cet IDE permet de tester en temps réel, l'appli sur un émulateur (l'ordinateur va donc simuler un appareil android) et donc voir si l'application fonctionne correctement ou pas.

- **Langages utilisées**

Comme évoqué ci-dessus, les contraintes du projet nous imposent à utiliser ces langages de programmation pour faire notre projet NAO, on utilise donc:

- Du **Python**, d'une part parce que c'est le langage qu'on utilisera pour faire la partie du NAO, c'est-à-dire les mouvements du NAO, la reconnaissance vocale, la géolocalisation du NAO et ainsi que le traitement des signaux, qui sont émis par le téléphone. D'autant plus qu'il s'agit là du langage qu'on utilise le plus pendant les cours de ce semestre.
- Du **Java**, pour créer le “backend” de notre application Android, c'est-à-dire la création des fonctionnalités de l'application (telle que la communication de l'utilisateur vers le NAO et ainsi que sa réception).
- Du **XML**, pour créer le “front-end” de notre application Android, c'est-à-dire les interfaces graphiques de l'application Android.

- **Outils et Framework utilisés**

Afin de nous faciliter la tâche durant le développement de notre projet, notre groupe, s'était mis d'accord pour utiliser ces outils suivants:

- **NaoQi**, un framework créé par SoftBank Robotics, il permet donc de gérer les fonctionnalités essentielles du NAO, telle que la vision, les mouvements, la reconnaissance vocale, les interactions avec le Nao mais aussi les communications avec les différents modules du NAO, lui permettant de coordonner ses actions. En effet, ce framework se révèle être utile puisque, toutes les fonctions qui permettent au NAO de pouvoir fonctionner correctement, y sont dessus, et que toute la documentation de ce framework se trouve sur le site internet du fabricant.
- **OpenCV**, est un framework créé initialement par Intel, mais licencié par Apache, il permet de traiter les images en temps réel. En effet, avec l'aide de OpenCV, on peut gérer les rendus graphiques de la caméra du NAO, ainsi que la perception de l'environnement et la reconnaissance des objets.
- **Physics Toolbox Suite**, il s'agit d'une collection d'outils logiciels disponible sur le Google Play Store. Cette application vise à exploiter les capteurs, et permet de faciliter les expérimentations avec la physique et la détection des mouvements, grâce à ça, on peut récupérer les coordonnées GPS, les données du gyroscope du NAO. On se sert de cette application pour pouvoir géolocaliser le NAO.

Réalisation

Organisation du dépôt git

Réaliseurs : Remy

Rédacteurs : Remy

Description

Avant la phase de réalisation, Remy a créé le dépôt git. Grâce au découpage des tâches fait durant la phase d'analyse, il a pu organiser les branches afin que tout le monde puisse réaliser les tâches qui leur ont été affectées. Les branches ont été créées de façon à suivre la hiérarchie du découpage des sous-tâches. Le dépôt a été séparé en deux dossiers : projet_android et projet_nao. Le premier répertoire contient le projet Android Studio et le deuxième contient tous les codes python qui seront exécutés sur le robot NAO.

Mise en place du projet Android Studio

Réaliseurs : Remy, Marius, Moïra, Lazar

Rédacteurs : Remy

Initialisation du robot Nao

Réaliseurs : Gabriel, Lazar

Rédacteurs : Gabriel

Description

Une fois le dépôt Git créé, il faut se connecter au robot Nao. Pour se connecter au Nao, il faut en premier lieu s'assurer qu'il est bien connecté au même réseau que la machine sur laquelle nous tentons la connexion. Les robots Nao n'ayant pas été réinitialisés lors du projet d'il y a deux ans, ils étaient déjà connectés au wifi de la salle. Cela nous a permis de pouvoir se connecter à la page internet du robot (disponible en tapant son IP dans une barre de recherche d'un navigateur, en ayant récupéré son IP en appuyant sur son bouton sur le torse). Une fois dessus, presque aucune fonctionnalité en dehors du changement de réseau, de langue, et de volume étaient accessibles, donc nous avons fait une connexion en ssh (via la commande dans une invite de commande: ssh nao@'adresseIP', le mot de passe étant "nao". Une fois une connexion établie nous avons pu récupérer et étudier le code des années précédentes laissé sur les Nao. Une fois ceci fait, nous avons tenté d'interagir avec l'API (Application Programming Interface ou « interface de programmation d'application ») de Nao sur Python sur nos machines. Le problème étant que pour cela nous devons utiliser la bibliothèque NaoQi, mais qui n'était pas détectée par Python puisqu'elle a été installée manuellement et n'était pas dans le dossier approprié. Pour contourner ce problème nous avons trouvé la solution de rajouter manuellement la bibliothèque au début de tous nos

codes via la ligne “sys.path.append("C:/pynaoqi/lib")”. Une fois notre environnement prêt nous avons commencé à interagir avec l’API de Nao afin de lui faire faire des actions basique tel que dire “Hello World”. Il n’a pas été compliqué de lui faire faire des actions plus complexes notamment grâce à la documentation (partiellement complète) d’Aldebaran http://doc.aldebaran.com/2-1/home_nao.html. Nous avons donc testé plus ou moins toutes les actions du Nao. Rapidement plusieurs problèmes sont apparus, notamment des fonctionnalités défectueuses (tel que Text To Speech ou l’accès à une caméra par exemple) et des problèmes de permissions (nous ne pouvions pas modifier les fichiers du Nao). Nous avons pris la décision de réinitialiser le NAO. Dû au manque de documentation sur internet et aux restrictions de nos ordinateurs de travail, il a été très compliqué de le faire. Je (Gabriel) suis tout de même parvenu à remettre les NAO en paramètre d’usine mais il s’est avéré qu’un bug est apparu, et que dès que nous installions une langue (le français) le robot une fois démarré, ne se rallumera jamais (son bouton central clignote ad vitam æternam).

Les difficultés rencontrées

Les principales difficultés ont été le manque de documentation même sur les sites des constructeurs, ainsi que les bug à répétition des Nao. Les 10 premières heures du projet ont très majoritairement rimé avec bug, recherche infructueuse sur internet et test en tout genre. De plus nous n'avons eu quasiment aucune aide des professeurs ou du département informatique (CCRI).

Mise en place du projet Android Studio

Réaliseurs : Remy, Marius, Moïra, Lazar

Rédacteurs : Remy

Description

Afin d’éviter les problèmes de compatibilité entre les différents développeurs de l’application Android, il a fallu créer le projet android et l’envoyer sur le dépôt Git afin que tout le monde puisse récupérer le même projet. Nous avons choisi de se mettre sur le SDK Platform en Android 7.0 (Nougat). Le SDK platform comprend la version du système d’exploitation Android, celle de l’API Android, les images systèmes pour les émulateurs et des outils et librairies pour le développement d’applications Android. Nous avons choisi Java pour gérer le back-end de l’application. Cette configuration est celle que nous avions utilisée en S4 avec Mme Keller pour apprendre le développement Android. Cette configuration permet aussi d’avoir une application utilisable sur la plupart des téléphones Android.

Les difficultés rencontrées

Au tout début, les premières configurations d’Android Studio sur les ordinateurs de l’IUT nous ont beaucoup ralenties dans le projet. Le guide de S4 pour la configuration était obsolète à certains endroits. Il a fallu bidouiller et solliciter le CCRI afin de pouvoir lancer Android Studio, les premières fois et pouvait prendre jusqu’à presque une heure. De plus, cette configuration était réinitialisée à chaque fois qu’on redémarrant les ordinateurs. Grâce au procédé de configuration mis sur le Discord et fait par Marius, la configuration d’Android Studio est passée aux alentours de 15 minutes (La configuration d’Android Studio est dans

les annexes). Pour éviter cette perte de temps de configuration, certains ont utilisé leur ordinateur personnel ou un disque dur avec Android Studio dessus. A la création du projet Android, au lieu de choisir Java, nous avions pris Kotlin. Nous pensions que le Kotlin était la même chose que Java mais avec des outils en plus. Il s'est avéré que le Kotlin même s'il venait de Java avait sa propre syntaxe qui était plus courte afin de faciliter la programmation. Comme nous n'avions jamais utilisé le kotlin, nous avons finalement décidé de re-créer le projet Android en Java.

Débogage du projet Android Studio

Réaliseurs : Remy, Marius, Gabriel

Rédacteurs : Remy, Marius

Description

En plus des problèmes de configuration d'Android Studio, nous avions des problèmes de connexion au robot avec Android Studio. Marius a identifié que les ordinateurs de l'IUT ne pouvaient pas communiquer avec les robots NAO en utilisant du code Java et aussi que la version Android Studio bloquait cette communication. En désactivant les pare-feu de l'ordinateur et en utilisant la version la plus récente d'Android Studio, nous avons réussi à régler ce problème.

Cette partie peut sembler rapide, mais cela a pris beaucoup de temps, notamment pour identifier les problèmes mais aussi pour convaincre le CCRI de désactiver le pare-feu de certaines machines ou d'installer la version plus récente d'Android studio (2023.1.1) sur certaines machines. Ainsi, afin de pouvoir travailler malgré cela, nous avons décidé d'utiliser Eclipse afin de coder en JAVA, notamment les serveurs de communication.

Programmation des interfaces de l'application sur Android Studio

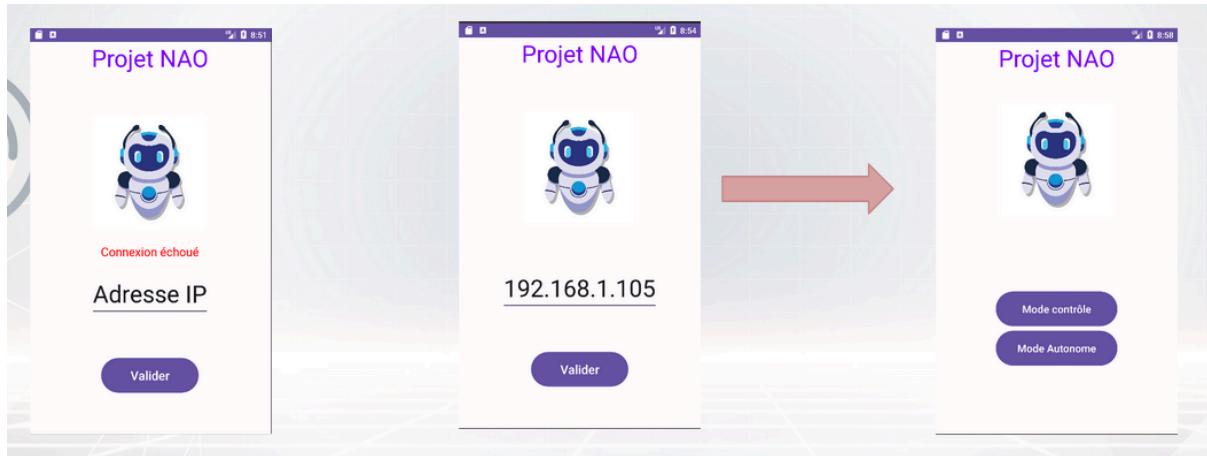
Réaliseurs : Remy

Rédacteurs : Remy

Description

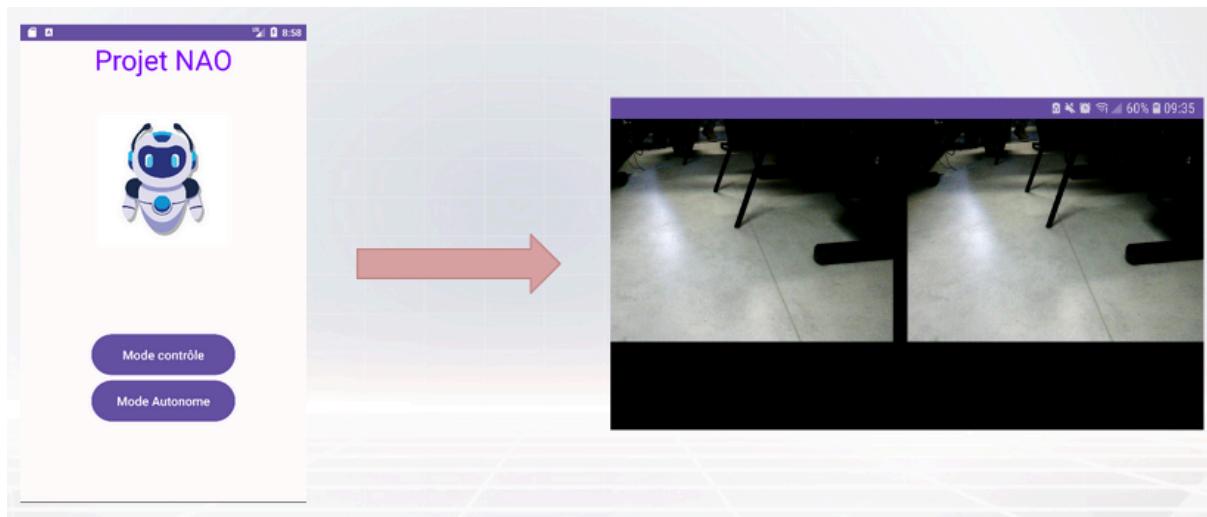
L'application se devait d'être agréable, facile à utiliser et adaptée à la majorité des téléphones possibles. Pour cela, les interfaces ont été programmées avec des ConstraintLayout. Les ConstraintLayout permettent de placer les éléments de l'activité en fonction des autres ce qui permet de ne pas se soucier de leur placement. Il est aussi possible de leur mettre des poids afin qu'elles prennent plus de place ou non.

La page de connexion



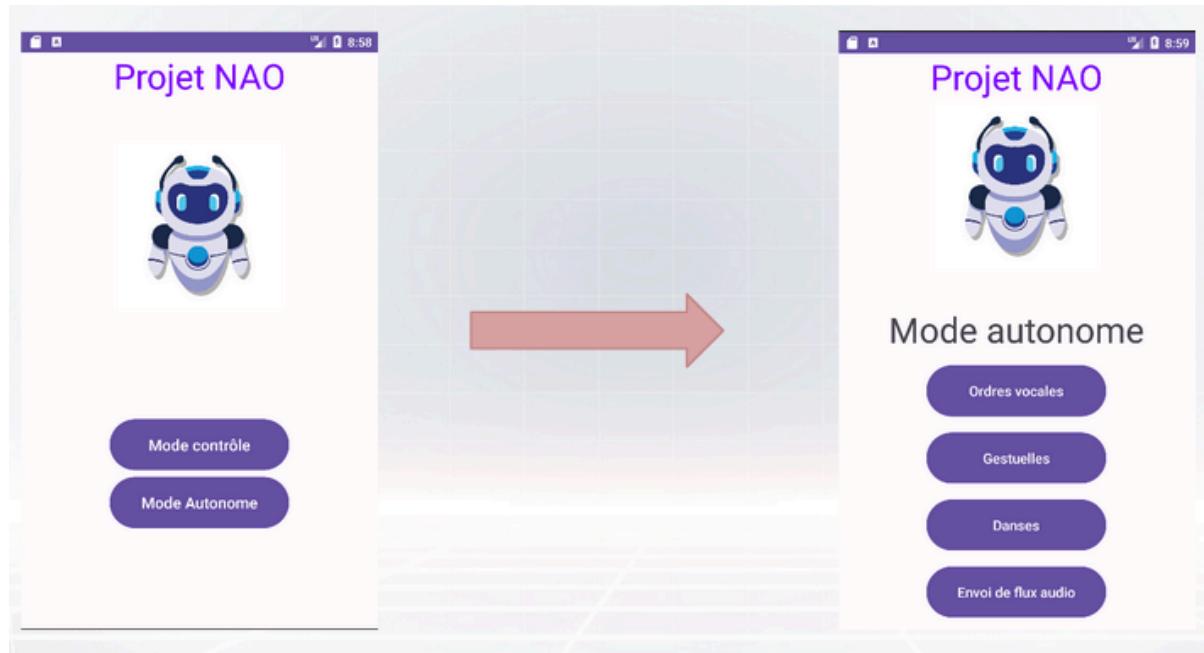
Un test de connexion au robot sur le port 80 est fait avant que l'utilisateur puisse accéder au menu. Si le test échoue, un message d'erreur est renvoyé sinon l'utilisateur peut accéder au menu. L'adresse IP sera passée entre les différentes activités grâce à un Intent. L'Intent permet de passer des variables entre chaque activité.

Le mode contrôle



Les vidéos sont affichées avec deux SurfaceView. Dans ces deux SurfaceView, deux zones de dessins sont créées (Canvas) afin d'y mettre les images successives de la vidéo qui seront en Bitmap.

Le mode autonome



Il était initialement prévu en fonctionnalités complémentaires qu'on puisse donner des ordres au robot, lui faire faire des gestuelles pré-faites, des danses ou encore faire jouer des fichiers audios. Ces commandes sont implémentées pour fonctionner avec la voix de l'utilisateur au lieu des boutons.

Récupération et envoi des données du gyroscope

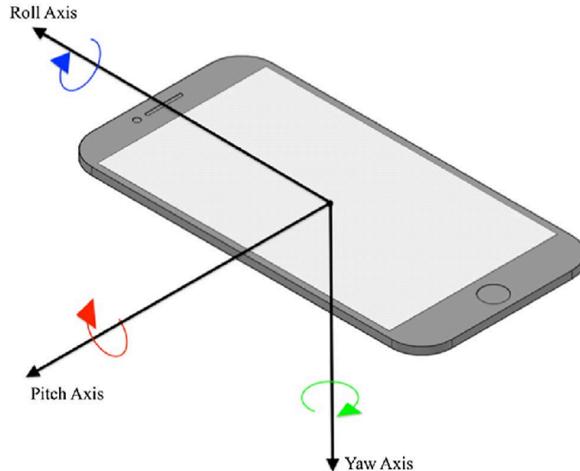
Réaliseurs : Lazar

Rédacteurs : Lazar

Description :

L'objectif de la partie gyroscope est de récupérer l'inclinaison du téléphone pour ensuite pouvoir envoyer ces données à Nao qui va synchroniser ses mouvements avec ceux du téléphone. Par exemple, si l'utilisateur dirige le téléphone vers la droite, Nao tournera sa tête vers la droite, et cela continuera pour toutes les directions.

Pour réaliser ceci, il faut tout d'abord accéder aux capteurs du téléphone (accéléromètre, gyroscope et magnétomètre), puis les traiter en passant par des formules mathématiques pour obtenir le pitch, le yaw et l'azimut. Enfin, ces données sont envoyées à un serveur distant.



Les sous tâches d'implémentation de cette logique dans une application Android :

Sous-tâche 1 : Initialisation des capteurs et des éléments d'interface utilisateur

Dans cette partie, les capteurs (gyroscope, accéléromètre, magnétomètre) sont initialisés, et les éléments d'interface utilisateur sont liés aux vues correspondantes dans le fichier de mise en page (layout XML).

```
// Gyroscope Sensor
gyroscopeSensor = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
if (gyroscopeSensor == null) {
    Toast.makeText(context, text: "Le gyroscope n'est pas accessible", Toast.LENGTH_SHORT).show();
    finish();
}

// Accelerometer Sensor
accelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
if (accelerometerSensor == null) {
    Toast.makeText(context, text: "L'accéléromètre n'est pas accessible", Toast.LENGTH_SHORT).show();
    finish();
}

// Magnetometer Sensor
magnetometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
if (magnetometerSensor == null) {
    Toast.makeText(context, text: "Le magnétomètre n'est pas accessible", Toast.LENGTH_SHORT).show();
    finish();
}

// Initialize handler for the main (UI) thread
handler = new Handler(Looper.getMainLooper());
}
```

```

resultat1TextView = findViewById(R.id.Titre1);
resultat2TextView = findViewById(R.id.Titre2);
resultat3TextView = findViewById(R.id.Titre3);

AccelerometerX = findViewById(R.id.ResultX);
AccelerometerY = findViewById(R.id.ResultY);
AccelerometerZ = findViewById(R.id.ResultZ);

pitchTextView = findViewById(R.id.MouvementX);
rollTextView = findViewById(R.id.MouvementY);
yawTextView = findViewById(R.id.MouvementZ);
positionTextView = findViewById(R.id.Position);
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

```

Sous-tâche 2 : Gestion du cycle de vie de l'application

Les méthodes suivantes gèrent l'enregistrement et la désinscription des écouteurs de capteurs en fonction du cycle de vie de l'application, et elles ferment également la connexion socket lors de la destruction de l'activité.

```

@Override
protected void onResume() {
    super.onResume();
    sensorManager.registerListener(listener, gyroscopeSensor, SensorManager.SENSOR_DELAY_NORMAL);
    sensorManager.registerListener(listener, accelerometerSensor, SensorManager.SENSOR_DELAY_NORMAL);
    sensorManager.registerListener(listener, magnetometerSensor, SensorManager.SENSOR_DELAY_NORMAL);
}

@Override
protected void onPause() {
    super.onPause();
    sensorManager.unregisterListener(this);
}

@Override
protected void onDestroy() {
    super.onDestroy();
    // Fermer les sockets une fois l'activité détruite
    if (socket != null && !socket.isClosed()) {
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

Sous-tâche 3 : Gestion des données des capteurs

La méthode suivante est appelée chaque fois que les données d'un capteur changent. Elle appelle ensuite des méthodes spécifiques pour gérer les données de chaque capteur, met à jour l'orientation et la position, puis envoie les données à un serveur distant via une connexion socket.

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {
        handleGyroscopeData(event.values);
    } else if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        handleAccelerometerData(event.values);
    } else if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
        handleMagnetometerData(event.values);
    }
    updateOrientation();
    calculatePosition();

    // Envoie au main(UI) thread
    handler.post(new Runnable() {
        @Override
        public void run() {
            sendSensorDataOverSocket();
        }
    });
}
```

Sous-tâche 4 : Traitement des données du gyroscope, de l'accéléromètre et du magnétomètre

Les méthodes suivantes traitent les données spécifiques de chaque capteur (gyroscope, accéléromètre, magnétomètre) en fonction de certains critères, puis mettent à jour les vues correspondantes dans l'interface utilisateur.

```
private void handleGyroscopeData(float[] values) {
    float limit = 0.1f;
    float gyroX = values[0];
    float gyroY = values[1];
    float gyroZ = values[2];

    if (Math.abs(gyroX) > limit || Math.abs(gyroY) > limit || Math.abs(gyroZ) > limit) {
        resultat1TextView.setText("Gyro X: " + gyroX);
        resultat2TextView.setText("Gyro Y: " + gyroY);
        resultat3TextView.setText("Gyro Z: " + gyroZ);
    }
}
```

```

private void handleAccelerometerData(float[] values) {
    float limit = 0.5f;
    float accX = values[0];
    float accY = values[1];
    float accZ = values[2];

    filteredAccelValues[0] = ALPHA * filteredAccelValues[0] + (1 - ALPHA) * accX;
    filteredAccelValues[1] = ALPHA * filteredAccelValues[1] + (1 - ALPHA) * accY;
    filteredAccelValues[2] = ALPHA * filteredAccelValues[2] + (1 - ALPHA) * accZ;

    if (Math.abs(filteredAccelValues[0]) > limit || Math.abs(filteredAccelValues[1]) > limit
        AccelerometerX.setText("Accelerometer X: " + filteredAccelValues[0]);
        AccelerometerY.setText("Accelerometer Y: " + filteredAccelValues[1]);
        AccelerometerZ.setText("Accelerometer Z: " + filteredAccelValues[2]);

        accelValues[0] = filteredAccelValues[0];
        accelValues[1] = filteredAccelValues[1];
        accelValues[2] = filteredAccelValues[2];
    }
}

1 usage
private void handleMagnetometerData(float[] values) {
    System.arraycopy(values, i: 0, magnetValues, i1: 0, i2: 3);
}

```

Sous-tâche 5 : Calcul de l'orientation(pitch, roll et azimuth) et envoi des données via la connexion socket

Les méthodes suivantes effectuent le calcul de l'orientation, puis envoient les données d'orientation à un serveur distant via une connexion socket. Pitch, roll et azimuth sont transformés en degré avant de les envoyer et leur valeurs vont de -180 à 180.

```

private void updateOrientation() {
    if (SensorManager.getRotationMatrix(rotationMatrix, null, accelValues, magnetValues)) {
        SensorManager.getOrientation(rotationMatrix, orientationValues);
        float azimuth = (float) Math.toDegrees(orientationValues[0]); // radians

        azimuth = (azimuth + 360) % 360;
        azimuth = (azimuth > 180) ? azimuth - 360 : azimuth;

        currentPitch = (float) Math.toDegrees(orientationValues[1]);
        currentRoll = (float) Math.toDegrees(orientationValues[2]);
        currentYaw = azimuth;

        pitchTextView.setText("Pitch: " + currentPitch);
        rollTextView.setText("Roll: " + currentRoll);
        yawTextView.setText("Azimuth: " + currentYaw);
    }
}

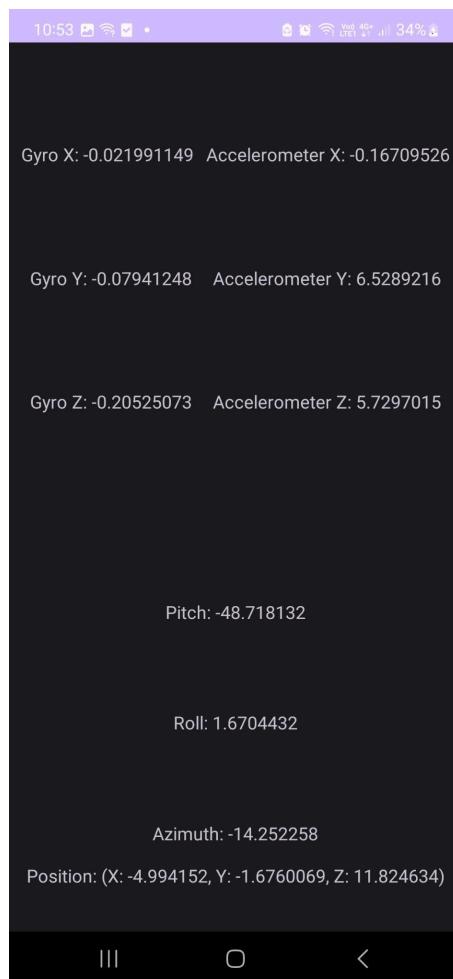
```

```

private void sendSensorDataOverSocket() {
    //Définition de string à envoyer
    String data = String.format("Pitch: %f, Roll: %f, Yaw: %f",
        currentPitch, currentRoll, currentYaw);
    new SocketTask().execute(data);
}

```

Le point de vue dans l'application Android :



Les problèmes rencontrés

J'ai rencontré plusieurs petits problèmes durant la réalisation de cette application. Parmi les plus importants :

1. Problèmes d'initialisation d'Android Studio sur certaines machines de la salle.
2. Problèmes de version d'Android Studio.
3. Problème avec le pare-feu qui empêche la connexion à un serveur distant.

Récupération et envoi des données de géolocalisation

Réaliseurs : Lazar

Rédacteurs : Lazar

Description

L'objectif de la géolocalisation est d'obtenir la position du téléphone afin de pouvoir ensuite synchroniser cette position avec Nao. Il existe plusieurs méthodes pour réaliser cela. Au début du projet, j'ai choisi de poursuivre la méthode de positionnement relatif du téléphone qui se fait de la manière suivante : à l'aide de l'accéléromètre et du gyroscope dans une application Android, le positionnement relatif peut être réalisé en utilisant les données des capteurs pour estimer les changements de position dans l'espace.

Voici une approche détaillée pour obtenir un positionnement relatif en utilisant l'accéléromètre et le gyroscope dans une application Android :

Sous-tâche 1. Acquisition des données des capteurs : (Gyroscope, accéléromètre et magnetometre)

Cette étape est déjà présente dans la partie gyroscope :

Sous-tâche 2. Intégration des données pour obtenir la position :

a. Filtrage des données :

Implémentation des techniques de filtrage (filtres passe-bas et filtres passe-haut) pour réduire le bruit dans les données des capteurs.

Les formules Mathématiques pour implémenter les filtres :

```

public void onSensorChanged(SensorEvent event)
{
    // alpha is calculated as t / (t + dT)
    // with t, the low-pass filter's time-constant
    // and dT, the event delivery rate

    final float alpha = 0.8;

    gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
    gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
    gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

    linear_acceleration[0] = event.values[0] - gravity[0];
    linear_acceleration[1] = event.values[1] - gravity[1];
    linear_acceleration[2] = event.values[2] - gravity[2];
}

```

L'implémentation filtres passe-bas de filtre bas sur l'accéléromètre :

```

private void handleAccelerometerData(float[] values) {
    float limit = 0.5f;
    float accX = values[0];
    float accY = values[1];
    float accZ = values[2];

    filteredAccelValues[0] = ALPHA * filteredAccelValues[0] + (1 - ALPHA) * accX;
    filteredAccelValues[1] = ALPHA * filteredAccelValues[1] + (1 - ALPHA) * accY;
    filteredAccelValues[2] = ALPHA * filteredAccelValues[2] + (1 - ALPHA) * accZ;
}

```

b. Intégration de l'accélération pour obtenir la vitesse :

Intégrez les données d'accélération par rapport au temps pour obtenir les vitesses au long des axes x, y et z

c. Intégration de la vitesse pour obtenir la position :

Intégrez les données de vitesse par rapport au temps pour obtenir les positions le long des axes x, y et z.

La méthode calculatePosition() répond à b et c :

```

private void calculatePosition() {
    long timestamp = System.currentTimeMillis();
    float dt = (timestamp - lastTimestamp) / 1000.0f; // Convert to seconds
    lastTimestamp = timestamp;

    float[] rotationMatrix = new float[9];
    float[] inclinationMatrix = new float[9];
    float[] gravity = new float[3];
    float[] geomagnetic = new float[3];

    // Recuperation de matrice rotation
    boolean success = SensorManager.getRotationMatrix(rotationMatrix, inclinationMatrix, accelValues, magnetValues);

    if (success) {
        // Ajustement de matrice rotation par rapport a l'emplacement de telephone
        float[] remappedRotationMatrix = new float[9];
        SensorManager.remapCoordinateSystem(rotationMatrix, SensorManager.AXIS_X, SensorManager.AXIS_Y, remappedRotationMatrix);

        // Obtenir l'orientation de telephone
        float[] orientationValues = new float[3];
        SensorManager.getOrientation(remappedRotationMatrix, orientationValues);

        // Obtenir l'orientation de telephone
        float[] orientationValues = new float[3];
        SensorManager.getOrientation(remappedRotationMatrix, orientationValues);

        // Faire une mise a jour de la position en fonction d'orientation
        float azimuth = (float) Math.toDegrees(orientationValues[0]);
        float pitch = (float) Math.toDegrees(orientationValues[1]);
        float roll = (float) Math.toDegrees(orientationValues[2]);

        // Faire une mise a jour de la position actuelle
        currentPosition[0] += Math.sin(Math.toRadians(azimuth)) * Math.cos(Math.toRadians(pitch)) * dt;
        currentPosition[1] += Math.sin(Math.toRadians(pitch)) * dt;
        currentPosition[2] += Math.cos(Math.toRadians(azimuth)) * Math.cos(Math.toRadians(pitch)) * dt;

        // Afficher sur l'écran
        positionTextView.setText("Position: (" +
            "X: " + (currentPosition[0]) + ", " +
            "Y: " + (currentPosition[1]) + ", " +
            "Z: " + (currentPosition[2]) + ")");
    }
}

```

Sous-tâche 3. Initialisation du point de départ :

Lorsque l'application est lancée, le point de départ est (0, 0, 0) pour les coordonnées x, y et z. Cela est considéré comme le point zéro de référence. En déplaçant le téléphone les coordonnées devront changer proportionnellement à la distance de manière continue.

Sous-tâche 4. Affichage de la position relative :

Affichez la position relative calculée par rapport au point de départ sur l'interface utilisateur de l'application.

```
// Afficher sur l'écran
positionTextView.setText("Position: (" +
    "X: " + currentPosition[0]) + ", " +
    "Y: " + currentPosition[1]) + ", " +
    "Z: " + currentPosition[2]) + ")");
}
```

L'approche décrite n'a pas été utilisé et a été remplacé par une autre méthode pour les raisons suivantes :

Cette approche comporte de grands risques, le principal étant l'accumulation des erreurs dans la position et le problème de gravité. Il est impossible d'enlever complètement le vecteur gravité de l'accéléromètre présent sur le téléphone.

Les filtres sont efficaces et éliminent entre 80% et 90% du vecteur gravité, mais cela varie énormément, ce qui contribue encore à l'accumulation d'erreurs. Enfin, avec cette méthode de suivi, la position du portable est précise seulement pendant les premières 10 à 20 secondes, puis, grâce à l'accumulation d'erreurs, elle devient complètement inutilisable, car même avec le téléphone qui ne bouge pas, les erreurs continuent à s'accumuler à l'infini.

C'est pour ces raisons précisées ici que j'ai décidé d'implémenter et d'utiliser une autre méthode. La nouvelle méthode combine le magnétomètre (boussole) et une fusion de capteurs pour calculer le nombre de pas réalisés par la personne en possession du téléphone. En fusionnant ces deux éléments, nous allons pouvoir utiliser la boussole pour déterminer la direction de téléphone, et le nombre de pas pour déterminer la distance parcouru, puis nous allons déplacer le robot Nao dans la même direction et d'un nombre de pas proportionnelle aux nombres de pas réalisé par l'utilisateur de téléphone.

Voici une approche détaillée pour obtenir la position en utilisant le compteur de pas et la boussole (La méthode actuellement utilisé pour déplacer Nao) :

Sous-tâche 1 : Initialisation des capteurs et des éléments d'interface utilisateur

Dans cette partie, les capteurs de compteur de pas et de détecteur de pas sont initialisés, et le TextView pour afficher le nombre de pas est lié à la vue correspondante dans le fichier de mise en page (layout) XML.

```
stepsTextView = findViewById(R.id.STEPS);

sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
stepCounterSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
stepDetectorSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR);

if (stepCounterSensor == null) {
    Log.d(tag: "StepCounterTest", msg: "onCreate: Step counter sensor not available on this device.");
    stepsTextView.setText("Step Counter Sensor not available on this device.");
}
```

Sous-tâche 2 : Gestion du cycle de vie de l'application

Les méthodes suivantes gèrent l'enregistrement et la désinscription des écouteurs de capteurs en fonction du cycle de vie de l'application.

```
@Override
protected void onResume() {
    super.onResume();
    Log.d(tag: "StepCounterTest", msg: "onResume: Entered onResume");

    if (stepCounterSensor != null) {
        Log.d(tag: "StepCounterTest", msg: "onResume: Step counter sensor is available.");
        Log.d(tag: "StepCounterTest", msg: "onResume: Registering listener for step counter sensor.");
        sensorManager.registerListener(listener: this, stepCounterSensor, SensorManager.SENSOR_DELAY_FASTEST);
        isStepCounterListenerRegistered = true;
    } else {
        Log.d(tag: "StepCounterTest", msg: "onResume: Step counter sensor is not available on this device.");
    }

    if (stepDetectorSensor != null) {
        Log.d(tag: "StepCounterTest", msg: "onResume: Step detector sensor is available.");
        Log.d(tag: "StepCounterTest", msg: "onResume: Registering listener for step detector sensor.");
        sensorManager.registerListener(listener: this, stepDetectorSensor, SensorManager.SENSOR_DELAY_FASTEST);
        isStepDetectorListenerRegistered = true;
    } else {
        Log.d(tag: "StepCounterTest", msg: "onResume: Step detector sensor is not available on this device.");
    }
}
```

```

@Override
protected void onPause() {
    super.onPause();
    Log.d( tag: "StepCounterTest", msg: "onPause: Entered onPause");

    if (isStepCounterListenerRegistered) {
        Log.d( tag: "StepCounterTest", msg: "onPause: Unregistering listener for step counter sensor.");
        sensorManager.unregisterListener( listener: this, stepCounterSensor);
        isStepCounterListenerRegistered = false;
    }

    if (isStepDetectorListenerRegistered) {
        Log.d( tag: "StepCounterTest", msg: "onPause: Unregistering listener for step detector sensor.");
        sensorManager.unregisterListener( listener: this, stepDetectorSensor);
        isStepDetectorListenerRegistered = false;
    }

    // Unregister other sensors if needed
}

```

Sous-tâche 3 : Gestion des données des capteurs

Cette méthode est appelée chaque fois que les données des capteurs changent. Elle vérifie le type de capteur et met à jour le nombre de pas en conséquence. Si le capteur de détecteur de pas détecte un pas, le nombre de pas est également incrémenté.

Le code implémente deux types de capteurs de pas : le capteur de compteur de pas (TYPE_STEP_COUNTER) qui calcule le nombre de pas totale depuis le dernier redémarrage du téléphone, et le capteur de détecteur de pas (TYPE_STEP_DETECTOR) qui s'exécute à chaque fois qu'un pas est détecté.

```

@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_STEP_COUNTER) {
        stepCount = (int) event.values[0];
        Log.d( tag: "StepCounterTest", msg: "onSensorChanged: Step count updated - " + stepCount);
    } else if (event.sensor.getType() == Sensor.TYPE_STEP_DETECTOR) {
        // Step detector event received
        Log.d( tag: "StepCounterTest", msg: "onSensorChanged: Step detected");
        stepCount++;
        counter++;
        stepsTextView.setText("Steps: " + stepCount + " Step detector " + counter);
    }
}

no usages
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
    Log.d( tag: "StepCounterTest", msg: "onAccuracyChanged: Sensor accuracy changed - " + accuracy);
}

```

Les problèmes rencontrés

Le plus grand problème rencontré durant cette étape a été l'accès et la permission concernant les détecteurs liés au corps et à la santé. Les compteurs de pas ne sont pas les capteurs traditionnels dans le sens où ils ne sont pas physiques ; ces compteurs sont des méthodes intégrées à l'intérieur des téléphones Android et utilisées pour les applications de santé telles que Samsung Health. Par conséquent, il faut obligatoirement ajouter des permissions dans le fichier XML, mais aussi les activer une fois que l'application est sur le téléphone physique. Il est donc impossible de réaliser les tests sur des téléphones virtuels.

Réception et envoi audio à partir du Téléphone

Rédacteur : Cordier Moïra

Réalisateur : Cordier Moïra

Rectifications

L'enregistrement vocal était envisagé pour transmettre à Nao. Pour cela, un bouton était nécessaire pour enregistrer le son puis la conversion en mp3 ou .WAV grâce à des encodages pour ainsi l'écouter. Tout cela se fait à l'aide des bibliothèques intégrées en java.

Tâches

Avant de commencer les tâches, Moïra essayé plusieurs façons de capturer l'audio, la première, c'est l'enregistrement audio mais ce n'était pas réalisable il fallait cependant appuyer sur un bouton à un certain moment pour lancer et stopper. La deuxième solution est le streaming Audio mais il s'est avéré plus compliqué et par manque de temps, nous n'avons pas pu choisir cette méthode. Enfin, la solution adoptée est le SpeechToText qui est plus réalisable et plus simple envoyée sur le serveur et de plus, cela faciliterait l'exécution des ordres au Nao.

Tâche Récupération de l'audio puis l'encoder en texte

Les API SpeechRecognizer et RecognizerIntent, permettent d'activer l'écoute depuis le microphone du téléphone puis de le son retranscrire en texte.

```
private void startContinuousSpeechRecognition() {
    // Configuration de la reconnaissance vocale
    Intent recognizerIntent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    recognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    recognizerIntent.putExtra(RecognizerIntent.EXTRA_PARTIAL_RESULTS, true);
    recognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, "fr-FR"); // Remplacez "fr-FR" par le code de langue souhaité
    // Commencez la reconnaissance vocale
    speechRecognizer.startListening(recognizerIntent);
}
```

Fonction qui permet de lancer la reconnaissance vocale

Tout d'abord il a fallu créer un “Intent” (classe permettant création d’activité) pour demander à l’application d’écouter l’utilisateur. Ensuite, il faut configurer les paramètres pour gérer la reconnaissance vocale, c'est-à-dire, on ajoute des “Extra” (méthode de la classe Intent) pour spécifier le modèle de langue à utiliser, activer la récupération de l’écoute en

texte et enfin préciser le modèle de langue détecté. Puis, une fois les paramètres configurés, on démarre la reconnaissance vocale.

Tâche Écouter en permanence

Ensuite avec RecognitionListener, qui permet de gérer l'écoute, grâce à des fonctions intégré, me permettra de configurer la manière de l'écoute, par exemple : la fonction onReadyForSpeech, va être appelé lorsque la reconnaissance vocale est prêt à recevoir des données audios, onEndOfSpeech est appelée à la fin, quand l'utilisateur a fini de parler. Ces deux fonctions vont permettre de gérer l'enregistrement en boucle jusqu'à qu'on le stop manuellement, la fonction onPartialResults va s'activer dès qu'un mot a été détectée pour qu'on ajoute le contenu de la reconnaissance vocale qui enregistre toutes les 1 secondes. Enfin la fonction onResults, va permettre de relancer la reconnaissance vocale dès que la fonction onEndOfSpeech est appelée.

Tâche Envoyer le contenu au serveur

Une fois les étapes précédentes réalisées, nous avons relié le texte détecté au système d'envoie de donnée du gyroscope, le String (chaîne de caractère) est légèrement formaté avant l'envoi de manière à filtrer les phrases envoyer. Uniquement les phrases contenant le mot "stop" sont envoyées au NAO pour être lu à voix haute par le code de Keryann (TTS).

Les difficultés

Les seules difficultés étaient de permettre l'écoute en continu puis d'éviter l'écoute sans qu'une fenêtre pop up s'affiche. Dans un premier temps, un bouton était initialisé pour lancer la reconnaissance vocale et une fenêtre pop up pour confirmer que la reconnaissance vocale était bien activé seulement que ceci générera pour le visionnage de la vidéo alors il a fallu changer la manière d'activer la reconnaissance vocal sans que la fenêtre affiche, après quelque recherche dans la documentation et avec l'aide de chatGPT, l'API SpeechRecognizer était la solution mais pas la bonne façon de l'utiliser. Et enfin, l'écoute en continu, le bon API nécessaire était le RecognitionListener, mais il s'est avéré plus complexe à comprendre et à l'utiliser.

Tâche Gestion du flux vidéo en direct entre le téléphone et le robot

Rédacteur : Marius, Loïck

Réalisateur : Marius, Loïck, Gabriel

Rectification

Pour recevoir le flux vidéo, il faut avoir créer un serveur qui va gérer en temps réel l'affichage du flux vidéo entre les yeux du Nao et l'affichage sur l'écran du téléphone.

Description

Pour pouvoir afficher la vidéo capturée par la caméra du Nao sur un téléphone en temps réel, il y a beaucoup d'étapes intermédiaires.

D'abord, il a fallu récupérer la vidéo image par image, l'image est ensuite encodée en fichier format jpg et elle est envoyé au serveur Java (d'abord hosté sur un serveur pour tester, puis ensuite intégrée au code de l'application), ou l'image est décodée et affichée.

Tâche Envoie du flux vidéo au serveur java

Nous avons premièrement passé du temps à déterminer quelle méthode serait la plus efficace. Nous avons envisagé d'enregistrer des fichiers vidéos en .avi et de les envoyer au serveur en boucle, mais cette solution n'a pas été retenue. A la place, nous avons récupéré des images à l'aide de la fonction 'getImageRemote' de la bibliothèque naoqi. Celle-ci nous renvoie un tableau avec les metadatas de l'image et un tableau de pixel. Dans un premiers temps, nous nous sommes contentés d'envoyer le tableau de pixel au serveur Java en passant par une socket, mais ce processus prenait beaucoup de temps dès que l'image était d'un peu bonne qualité. Nous avons donc encodé le tableau de pixel en une image format .jpg, afin qu'elle soit compresser et donc que l'envoie par le socket ne dure pas trop longtemps, nous permettant de concilier fluidité et qualité d'image.

Tâche Création de serveurs Java (Eclipse) de réception et affichage des images

Dans un premier temps, nous avons réalisé sur Eclipse, car Android studio ne fonctionnait pas à ce moment-là, un programme de traitement et affichage d'images. Nous avons donc repris les cours de Multimédia de M. Ravenet puisque ces derniers traitaient des images sous forme de tableau de pixel.

Nous avons donc mis en place quelques fonctions de formatage utilisant des expressions régulières, afin de traiter le tableau de pixel reçu dans un format qui correspondait à celui demandé par un JFrame (fenêtre java permettant, entre autres, d'afficher des images). Pour tester notre programme, nous avons converti quelques images en tableau de pixels et avons joué une animation en pixel-art dans un JFrame.



(animation de 8 frames en pixel-art) source : Marius PIRIS

```

1 ["pixels": [[[0, 0, 0, 0], [0, 0, 0, 0], [
0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [
0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [
0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [
0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [
0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [
0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [
0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [
0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]], [
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0],
[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0],
[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0],
[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0],
[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0],
[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0],
[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0],
[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]]

```

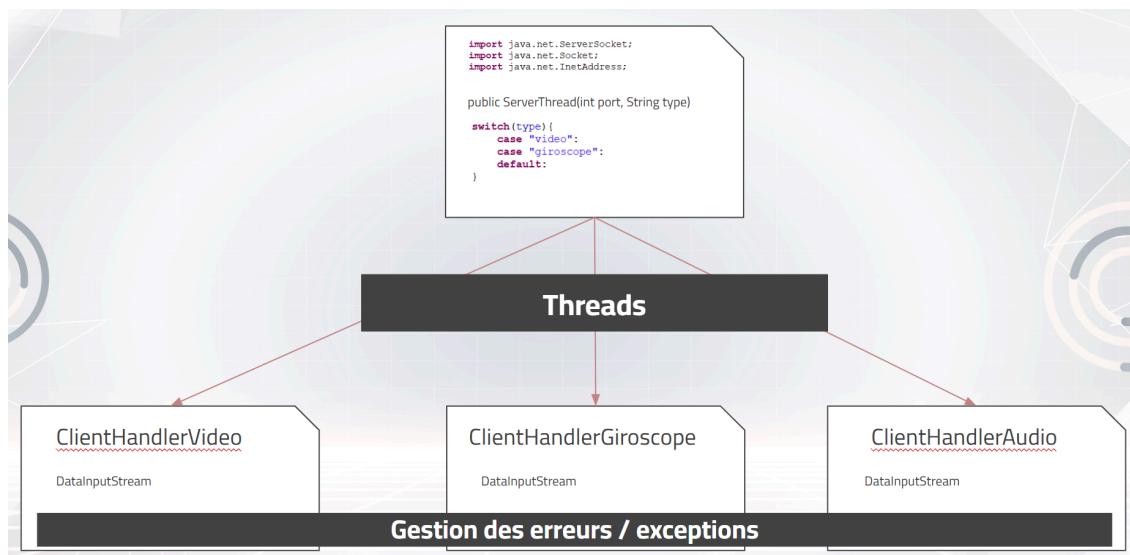
(tableau de pixel correspondant à une image) source : Marius PIRIS

Ensuite, nous avons mis en place un système de serveur-client avec des **sockets** (bibliothèque java et python) qui permettent de faire un “pont” entre le robot NAO et le programme d'affichage d'image Eclipse.

Enfin, nous avons tout d'abord mis en place un flux en utilisant un BufferedReader, mais ce dernier n'était pas très efficace et relativement lent, nous avons donc décidé d'utiliser une DataInputStream qui a un fonctionnement similaire à un BufferedReader (donc qui permet de recevoir des données sous forme de flux) mais cette fois ci en binaire, donc bien plus rapide et bien moins gourmand en mémoire.

Tâche Création de serveurs Java (Eclipse) d'envoie de donnée vers nao

Nous savions dès le départ que plusieurs serveurs devraient tourner en simultanés : un pour gérer l'échange de donnée audio (du robot vers le téléphone), un pour gérer l'échange de donnée en provenance du téléphone vers le NAO (comme les données du gyroscope ou les données issus du “speechToText” (le son capté par le micro du téléphone est envoyé sous forme de chaîne de caractère au robot), et un autre pour gérer la vidéo. Or, faire tourner plusieurs serveurs en simultanés peut être très gourmand en ressource, nous avons donc décidé de faire tourner ces serveurs sur le téléphone au lieu du robot afin de ne pas trop utiliser la mémoire du robot qui serait utilisé pour les déplacements. Ensuite, il nous fallait une architecture solide et facilement maintenable pour les serveurs sur le téléphone, nous avons donc décidé de faire plusieurs classes différentes en fonction du type de client connecté, par exemple la classe “ClientHandlerGyroscope” qui permet de gérer les données issues du gyroscope et de les envoyer au NAO.



(schéma explicatif des serveurs java) source : *Marius PIRIS*

Les difficultés

Pour ce qui est de récupérer les images, la difficulté majeure à été de trouver comment faire fonctionner les caméras, car nous avons passé du temps à déchiffrer la documentation, parfois bizarrement organisée ou peu claire. Il a donc fallu faire beaucoup de tests avant de réussir à récupérer une image non-vide de la caméra du robot. Nous n'avons d'ailleurs jamais réussi à faire fonctionner la caméra située sur le front du robot malgré nos efforts, nous avons donc utiliser la caméra située au niveau de la bouche du robot, ce qui rends donc la vision dans le casque VR un peu moins réaliste, mais toujours suffisamment pour remplir le cahier des charges.

Programmation du TextToSpeech

Rédacteur : Keryann

Réaliseurs : Keryan, Haolong

Rectifications

Nous n'avons pas modifier d'autres aspects de la voix, tels que le style de voix, la prononciation personnalisée, les pauses et délais car il fallait vérifier la disponibilité des nouvelles voix puis les télécharger nous-même depuis SoftBank Robotics et les installer sur le robot NAO sauf que nous n'avons pas les droits pour modifier la langue de NAO à cause de sa version.

Description

Afin de permettre au robot NAO de prononcer des phrases à partir de la saisie de l'utilisateur, un script Python a été développé en utilisant la bibliothèque NAOqi. Ce processus implique plusieurs étapes.

Dans un premier temps, le script importe les modules nécessaires pour la gestion du temps et les fonctionnalités système. Il établit également l'accès aux fonctionnalités de NAOqi. Ensuite, le script se connecte au robot NAO en spécifiant son adresse IP et son port. Une instance est créée pour contrôler la synthèse vocale du robot.

Une fonction est définie pour configurer certains paramètres de la synthèse vocale, tels que la vitesse, le changement de hauteur tonale et le volume.

Par la suite, une autre fonction permet à l'utilisateur d'entrer une phrase à l'aide de la fonction . La phrase est ensuite transmise à NAO grâce à la méthode `say` de l'interface `ALTextToSpeech` pour qu'il la prononce.

Enfin, le script vérifie si le programme est exécuté en tant que programme principal. Si tel est le cas, il appelle la fonction `dire_phrase()`, offrant ainsi à l'utilisateur la possibilité d'interagir en entrant une phrase que NAO prononcera.

Une fois ceci réalisé, nous avons configuré la vitesse, le ton, et le volume du robot.

Les difficultés rencontrées

Nous avons rencontré des problèmes avec la bibliothèque NAOqi car nous utilisions la fonction input de version 3.11 de Python or pour faire parler NAO cette fonction devait être raw_input car nous devions utiliser Python 2.7.

```
PS C:\Users\kdubocq> & c:/Python27/python.exe c:/Users/kdubocq/Downloads/test.py
[W] 1705419554.632341 10656 qi.path.sdklayout: No Application was created, trying to deduce paths
[I] 1705419554.637671 10656 qimessaging.session: Session listener created on tcp://0.0.0.0:0
[I] 1705419554.802760 10656 qimessaging.transportserver: TransportServer will listen on: tcp://172.16.0.149:50661
[I] 1705419554.802760 10656 qimessaging.transportserver: TransportServer will listen on: tcp://192.168.56.1:50661
[I] 1705419554.807426 10656 qimessaging.transportserver: TransportServer will listen on: tcp://127.0.0.1:50661
Entrez la phrase que vous voulez que NAO dise : Bonjour ! Je suis NAO ton robot préféré !
PS C:\Users\kdubocq>
```

Affichage de ce qui se passe dans la console quand on lance le code

Réception et envoi de l'audio à partir du robot NAO

Rédacteur : Marius

Réalisateur : Marius, Keryann et Haolong

Rectifications

L'API de NAO ne permet pas d'envoyer directement un flux de données provenant du micro vers un serveur externe.

Tâches Récupération du son que capte le NAO

Cette tâche fut particulièrement difficile, la documentation de NAOqi n'est pas complète et contient même des erreurs par endroit, à cause de cela, Keryann et Haolong n'ont pas réussi cette partie.

Voici quelques erreurs identifiées :

- la fonction getFrontMicEnergy() : cette dernière récupère un signal électrique en sortie du microphone, or lors de la lecture de ce signal, le son reçu est incompréhensible. Nous doutons qu'il faudrait faire une conversion mais malheureusement nous ne savons pas laquelle et nous manquons de temps.
- la fonction setClientPreferences() est sensé pouvoir établir un flux de son entre un client et le robot, mais cette fonction prend en paramètre le nom d'un service (par exemple "ALAudioDevice"), mais les services requis par cette fonction sont inconnus, ils ne sont pas disponibles dans la documentation, ni sur les forums et encore moins sur chatGPT.
- les fonctions startMicrophonesRecording() et stopMicrophonesRecording() permettent

d'enregistrer des fichiers audio au format .wav uniquement. Donc ils ne permettent pas d'envoyer des flux audio continue.

Il ne restait plus beaucoup de jours alors Marius et Gaetan de l'autre groupe ont décidé de travailler ensemble sur cette tâche afin de trouver une solution.

La première solution envisagée fut d'envoyer exclusivement la fin du fichier .wav pendant que ce dernier était en train d'être écrit par les fonctions startMicrophonesRecording() et stopMicrophonesRecording() afin de créer un flux continu.

Problèmes rencontrés :

- Nous avions fait en sorte d'envoyer la fin du fichier toutes les quelques secondes, mais l'écriture du fichier .wav était irrégulière par conséquent il y avait beaucoup de latences entre les différents envois. De plus, nous avions des problèmes de lecture car l'en-tête du fichier changeait à chaque nouvel enregistrement, ce qui posait des problèmes de lecture et de compatibilité.

Donc nous avons décidé de faire un nouvel enregistrement toutes les 1 secondes et de l'envoyer. Cela cause certes un problème de léger délai (1 seconde), mais cela résout tous nos autres problèmes.

Tâches Envoie du son que capte le NAO

Une fois le son envoyé, nous avons dû faire face à divers autres problèmes afin de récupérer et de lire ce son. Dans la documentation NAOqi, il est écrit que l'enregistrement serait en 16000 Hz. Or ce n'est pas le cas, nous avons passé énormément de temps à essayer de comprendre pourquoi le son n'était pas lisible par un humain, jusqu'à ce que nous comprenions que le son était ralenti. Nous avons donc compris que nous devions augmenter la fréquence afin d'augmenter la rapidité de lecture. Nous avons fait plusieurs tests et nous sommes parvenus à la conclusion que l'enregistrement était en 96000 Hz.

Après cela, un dernier problème nous faisait face, de temps en temps le NAO ralentissait et s'arrêtait d'envoyer les fichiers. C'est à cause de divers problèmes de réseau et de mémoire, pour contourner ce problème, nous avons décidé de très légèrement augmenter la fréquence afin que les sons soient joués plus rapidement que ce qu'ils sont enregistré. Ainsi, si la lecture prend du retard, elle est rattrapée petit à petit sans que cela soit vraiment différent pour des humains.

Gestion du déplacement des mains

Rédacteurs : Tiancheng

Réaliseurs : Tiancheng

Rectification

Contrôler différentes articulations pour qu'elles tournent à différents angles à l'aide d'un code python pendant un certain temps afin de faire exécuter à nao une action spécifique.

Sous tâche 1

Connectez à nao avec python 2.7 et testez avec le code suivant

```
connexion = ALProxy("ALTextToSpeech", nao_ip, nao_port)  
  
connexion.say("Bonjour")
```

Sous tâche 2

Contrôle des mouvements en contrôlant l'angle et la durée des joints

Par exemple pour le bras droite :

"RShoulderPitch", "RShoulderRoll", "RElbowYaw", "RElbowRoll", "RWristYaw", "RHand" sont les joints du bras droite

```
def move_arm(self, angles_dict, duration):  
    for joint_name, angle in angles_dict.items():  
        self.set_joint_angles(joint_name, angle, duration)
```

Vous pouvez régler librement l'angle de chaque joint, l'angle et la durée.

Mais il faut beaucoup d'essais pour obtenir le résultat souhaité.

Sous tâche 3

Mettre en place des actions préprogrammées, telles que faire un initialisation, faire un signe de la main, se gratter la tête, célébrer , agiter, etc.

```
def initialize_right_arm(self): ...  
  
def initialize_left_arm(self): ...  
  
def waving_arm(self, n): ...  
  
def waving_both_arm(self, n): ...  
  
def dontUnderstand_arm(self, n): ...  
  
def celebrate(self): ...
```

Sous tâche 4

Alignez le format du code sur celui de vos camarades de classe pour faciliter l'intégration et les tests ultérieurs, par exemple en définissant le bras comme un bras de classe et en plaçant les articulations, les angles, ALMotion, etc. dans la classe.

```
class Arm:  
    def __init__(self, arm_name, ip_nao, port_nao):  
        self.arm_name = arm_name  
        self.joint_names = []  
        self.motion_proxy = ALProxy("ALMotion", ip_nao, port_nao)
```

Sous tâche 5

Optimiser le code, par exemple, le code original était gaucher et droitier séparément, ce qui est très peu pratique à programmer et double la quantité de code. À la suggestion de Léo, j'ai modifié le code pour qu'il décide de déplacer les articulations de la main gauche ou de la main droite en fonction du nom de la classe Arm. La seule chose à noter est que lorsque l'on contrôle la main gauche et la main droite, si l'on doit faire le même mouvement, une partie de l'angle est inversée, sauf pour ShoulderPitch, pour effectuer le même mouvement dans la direction opposée, ajoutez "-"(1 à -1).

Modifier le code précédent

```
def waving_right_arm(self, motion_proxy, n): ...  
  
def waving_left_arm(self, motion_proxy, n): ...  
  
def waving_both_arm(self, motion_proxy, n): ...  
  
def dontUnderstand_right_arm(self,motion_proxy,n): ...  
  
def dontUnderstand_left_arm(self,motion_proxy,n): ...
```

```
def waving_arm(self, n): ...  
  
def waving_both_arm(self, n): ...  
  
def dontUnderstand_arm(self,n): ...
```

Les difficultés rencontrées

Parfois, les mouvements de nao ne sont pas satisfaisants et nécessitent plusieurs tests.

Il peut être très difficile de convertir le format du code

Gestion du déplacement des Pieds

Rédacteur : PHAM Léo

Réalisateur : PHAM Léo

Rectification

Écrire des fonctions permettant de pouvoir faire marcher (au sens littéral) le NAO, et ainsi que le fait de reculer, tourner, lever le pied ou encore le fait de lui dire de se lever.

Sous tâche 1 : La création d'une classe pied et ses dérivées

Afin de permettre la gestion des deux pieds, Léo devait alors créer une classe pied (Leg) avec différents attributs tels que les coordonnées X, Y, ainsi que l'angle θ , mais aussi, les "pas", les angles, etc... Par ailleurs, le constructeur devra alors prendre un nom de pied mais aussi une adresse IP et ainsi qu'un port.

```
class Leg(object):
    def __init__(self, leg_name, ip_nao, port_nao):
        self.leg_name = leg_name
        self.X = 0
        self.Y = 0
        self.Theta = 0
        self.foot_steps = [[self.X, self.Y, self.Theta]]
        self.time_list = [0.1] # Effectue un mouvement chaque 0.1 secondes
        self.fractionMaxSpeed = [0.0] # La vitesse de base, qu'on va devoir changer (0.0 = inactif)
        self.clear_existing = False # Pour pas qu'il revient à sa position de base
        self.stiffnesses = [1.0]
        self.motion_proxy = ALProxy("ALMotion", ip_nao, port_nao)

        # Pour lever le pied
        self.leg_parts_Pitch = ["HipPitch", "KneePitch", "AnklePitch"]
        self.leg_parts_Roll = ["HipRoll", "AnkleRoll"]
        self.anglesPitch = [0.0, 0.0, 0.0]
        self.anglesRoll = [0.0, 0.0]
```

Ensuite, on devait créer des fonctions dans la classe pour un pied afin qu'il puisse avoir les angles corrects:

```

def set_Leg_Parts_Pitch(self):
    if self.leg_name == "RLeg":
        for i in range(len(self.leg_parts_Pitch)):
            self.leg_parts_Pitch[i] = "R" + self.leg_parts_Pitch[i]
    elif self.leg_name == "LLeg":
        for i in range(len(self.leg_parts_Pitch)):
            self.leg_parts_Pitch[i] = "L" + self.leg_parts_Pitch[i]

def set_Leg_Parts_Roll(self):
    if self.leg_name == "RLeg":
        for i in range(len(self.leg_parts_Roll)):
            self.leg_parts_Roll[i] = "R" + self.leg_parts_Roll[i]
    elif self.leg_name == "LLeg":
        for i in range(len(self.leg_parts_Pitch)):
            self.leg_parts_Roll[i] = "L" + self.leg_parts_Roll[i]

```

Quelques setters afin de pouvoir définir à l'extérieur de la classe des attributs de notre objet Leg et ainsi que des getters pour récupérer des valeurs dans les attributs de classe. Cependant, on a dû créer des fonctions permettant de pouvoir exécuter des fonctions qui sont dans l'API de NaoQi, de la classe ALMotion afin que le pied puisse par exemple, avancer, lever le pied, durcir le pied, etc...

```

def execute_foot_steps(self, motion_proxy):
    leg = [self.leg_name]

    steps = [list(map(float, step)) for step in self.foot_steps]
    motion_proxy.setFootSteps(leg, steps, self.time_list, self.clear_existing)

```

Puisqu'il s'agit de la deux pieds un pied gauche et un pied droit, on a donc fait deux classes (RightLeg, LeftLeg) dérivées de la classe Leg afin de pouvoir gérer ces deux pieds.

```

class LeftLeg(Leg):
    def __init__(self, ip, port):
        super(LeftLeg, self).__init__("LLeg", ip, port)

class RightLeg(Leg):
    def __init__(self, ip, port):
        super(RightLeg, self).__init__("RLeg", ip, port)

```

Et enfin, vient la réalisation des fonctions, moveLeg(), moveLegLateral() et TurnLeg() (qui n'est malheureusement jamais utilisé). Il s'agit de la pour le coup des fonctions importantes, car cela lui permettait de pouvoir utiliser les fonctions de setters afin de pouvoir assigner la vitesse à laquelle le NAO va avancer, ou encore la distance que le NAO devra faire pour pouvoir déposer son pied par terre...

```

def MoveLeg(self, distance):
    self.set_fraction_max_speed(0.3) # Pour pas que ce soit trop vite
    self.set_foot_steps(distance, 0, 0)
    self.execute_foot_steps(self.motion_proxy)

def MoveLegLateral(self, distance):
    self.set_fraction_max_speed(0.3)
    self.set_foot_steps(0, distance, 0)
    self.execute_foot_steps(self.motion_proxy)

def TurnLeg(self, theta):
    self.set_fraction_max_speed(0.3)
    self.set_foot_steps(0, 0, theta)
    self.execute_foot_steps(self.motion_proxy)

```

Petite parenthèse, on l'a mis à 0.3 en vitesse, car s'il mettait au-delà de cette valeur, le NAO ne serait plus trop stable et donc tomberait dû à son instabilité.

On a fait le choix d'utiliser la fonction setFootSteps() du Framework naoQi contrairement à moveTo(), tout d'abord elle ne fonctionne que sur un seul pied, par exemple s'il devait faire un pas de danse avec le NAO, il faudrait juste qu'on puisse pouvoir utiliser cette fonction par rapport à moveTo qui déplace les deux jambes. Ensuite, setFootSteps est modulaire, en effet, on peut très bien modifier la vitesse à laquelle le NAO pourra de déplacer, le temps qu'il devra mettre pour avancer, la distance à laquelle le NAO devra effectuer pour déposer le pied par terre, avec moveTo, on ne peut que faire d'avancer ou reculer, changer d'angle sans trop modifier quoi que ce soit.

Sous tâche 2 : La création des fonctions utilisant la classe Leg et ses dérivées

Avant de créer ces fonctions, il faudrait initialiser nos pieds gauche et droit, pour cela il faudrait juste créer une instance pied_gauche et pied_droit (bon ici ce sera left leg and right_leg).

La fonction MoveForwardOrBackward (Avancer ou reculer) prend en argument, un nombre de pas et une direction. Donc il répétera pour le nombre de pas, l'appel de la fonction MoveLeg des deux pieds, à une distance de 0.2m, si la direction est "front" (soit 20 centimètres, en effet, avec notre groupe, on s'était décidé que 20cm = 1 pas). Par ailleurs, si on a choisi de reculer, c'est-à-dire "back" au lieu de "front", il bougera de -0.2m (il reculera de 20 cm). C'est aussi le même principe pour la fonction moveLateral, mais au lieu d'avoir une direction "front" ou "back", il aura donc une direction "left", "right", mais aussi il va pouvoir appeler la fonction MoveLegLateral pour les deux pieds.

Enfin pour la fonction LiftLeg, il s'agit de la une fonction pour lever un pied (traction littérale). Il va donc lever un pied, via la fonction de classe lift_leg (on est désolé pour la confusion), afin de pouvoir modifier nos angles des joints du NAO afin qu'il puisse lever le pied comme on lui demander de faire. Ensuite, on renforce le pied opposé au pied qui s'était levé, afin que le NAO puisse avoir un équilibre sans tomber. Une fois que le NAO a fini de lever son pied, il va affaiblir son pied opposé et donc baisser le pied qui était du coup levé.



Voici, tout le résultat en code:

```

def MoveForwardOrBackward(nb_pas, sens):
    for j in range(nb_pas):
        if sens == "back":
            left_leg.MoveLeg(-0.2)
            right_leg.MoveLeg(-0.2)
        elif sens == "front":
            left_leg.MoveLeg(0.2)
            right_leg.MoveLeg(0.2)

def MoveLateral(nb_pas, sens):
    for j in range(nb_pas):
        if sens == "right":
            left_leg.MoveLegLateral(-0.2)
            right_leg.MoveLegLateral(-0.2)
        elif sens == "left":
            left_leg.MoveLegLateral(0.2)
            right_leg.MoveLegLateral(0.2)

def MoveOnlyTheta(theta): #Pour les deux pieds
    motion = right_leg.get_Motion_Proxy()
    motion.moveTo(0, 0, theta)

def LiftLeg(angle, choice):
    theta = math.radians(angle)

    if choice == "right":
        left_leg.lift_leg(theta, left_leg.get_Motion_Proxy())
        time.sleep(2)
        right_leg.setStiffnesses([1.0] * len(right_leg.get_leg_parts_Pitch() + right_leg.get_leg_parts_Roll()))
        right_leg.enableStiffness(right_leg.get_Motion_Proxy())
        time.sleep(2)
        right_leg.disableStiffness(right_leg.get_Motion_Proxy())
        left_leg.lower_leg(left_leg.get_Motion_Proxy())

    if choice == "left":
        right_leg.lift_leg(theta, right_leg.get_Motion_Proxy())
        time.sleep(2)
        left_leg.setStiffnesses([1.0] * len(left_leg.get_leg_parts_Pitch() + left_leg.get_leg_parts_Roll()))
        left_leg.enableStiffness(left_leg.get_Motion_Proxy())
        time.sleep(2)
        left_leg.disableStiffness(left_leg.get_Motion_Proxy())
        right_leg.lower_leg(right_leg.get_Motion_Proxy())

```

Petite précision supplémentaire, on a du faire une fonction MoveOnlyTheta, afin qu'il puisse pouvoir changer d'angle, si jamais on lui dit de tourner à un certain angle ou encore s'il on lui dit de marcher en diagonal, pour le coup, on a utilisé la fonction moveTo() de l'API de NaoQi, toujours de la même classe ALMotion, car on avait eu un problème dans la fonction de classe TurnLeg par Léo, il ne tournait pas assez correctement bien (quand on lui dit de tourner d'1 radian c'est à dire environ 57 degrées, il ne tourne pas plus de 15 degrées). on a donc pris la solution du moveTo qui lui permettait alors de bouger les 2 pieds du NAO.

Tester les mouvements du NAO

Rédacteurs : PHAM Léo, JIN Tiansheng

Réaliseurs : PHAM Léo, JIN Tiansheng

Rectification

Tester les mouvements de nao après chaque modification du code pour s'assurer que nao peut effectuer le mouvement comme prévu et ne tombe pas.

Sous tâche 1

On a donc réalisé des tests pour le mouvement des mains pour le NAO, afin de pouvoir décider, si une fonction est exécutée sans erreurs, on a pu fixer des erreurs.

Sous tâche 2

On a donc réalisé des tests pour le mouvement des pieds pour le NAO, afin de pouvoir décider, si une fonction est exécutée sans erreurs, on a pu fixer des erreurs.

Les difficultés rencontrées

Certains des mouvements du NAO sont plutôt satisfaisants dans le sens ou tout s'était passé comme prévu. Puis d'autres mouvements, qui nous laissent plutôt à désirer... dans le sens que tous les mouvements ne pas trop bien exécuté, notamment les nombreux tests du lever de pied du NAO qui s'est avéré comme étant éprouvant dans le sens, qu'on avait peur de le détruire avec le NAO qui chute assez souvent.

D'autant plus que l'exécution simultanée de plusieurs actions entraîne une déformation de l'ensemble des actions en raison de l'inachèvement de l'action précédente.

Réalisation des mouvements de la tête

Rédacteurs : Gautier Gabriel

Réaliseurs : Gautier Gabriel

Sous tâche 1:

La première étape a été de bouger la tête, de haut en bas et de droite à gauche, ce qui fut assez rapide. Le problème vite rencontré sont les suivants: entre chaque demande de mouvement une légère pause se fait ce qui nullifie la fluidité du mouvement. Le deuxième problème rencontré est que Nao ne peut pas exécuter de mouvement trop rapide. Pour de haut en bas sa vitesse maximum est de 7.1 rad/s et pour de droite à gauche 8.1 rad/s. Cela veut dire qu'il faut faire attention à de potentiels mouvements impossibles. Une des pistes explorées a été le fait qu'on puisse fournir deux tableaux en entrée pour un mouvement de Nao, avec un tableau pour une ligne de temps et un tableau pour une liste d'angles. Cela a pour effet de ne créer aucun délai entre les différents mouvements à faire. Cette solution est parfaite dans le cas où nous avons une liste de mouvements prédéfinis (comme voir ci-dessous: (exemple de code pour bouger la tête))



```

from naoqi import ALProxy

motion_proxy = ALProxy("ALMotion", "127.0.0.1", 9559)

# Première façon
motion_proxy.angleInterpolation(["HeadPitch"], radAngle, duration, True)
motion_proxy.angleInterpolation(["HeadYaw"], radAngle, duration, True)

# Deuxième façon
motion_proxy.angleInterpolation(["HeadPitch", "HeadYaw"], [radAnglePitch, radAngleYaw],
                                [duration, duration], True)

# Troisième façon
duration = [ 1, 2 , 3, 4, 5]
angle = [ 1, 0.5, 0, -0.5, -1]
motion_proxy.angleInterpolation(["HeadPitch", "HeadYaw"], [angle, angle], [duration, duration], True)

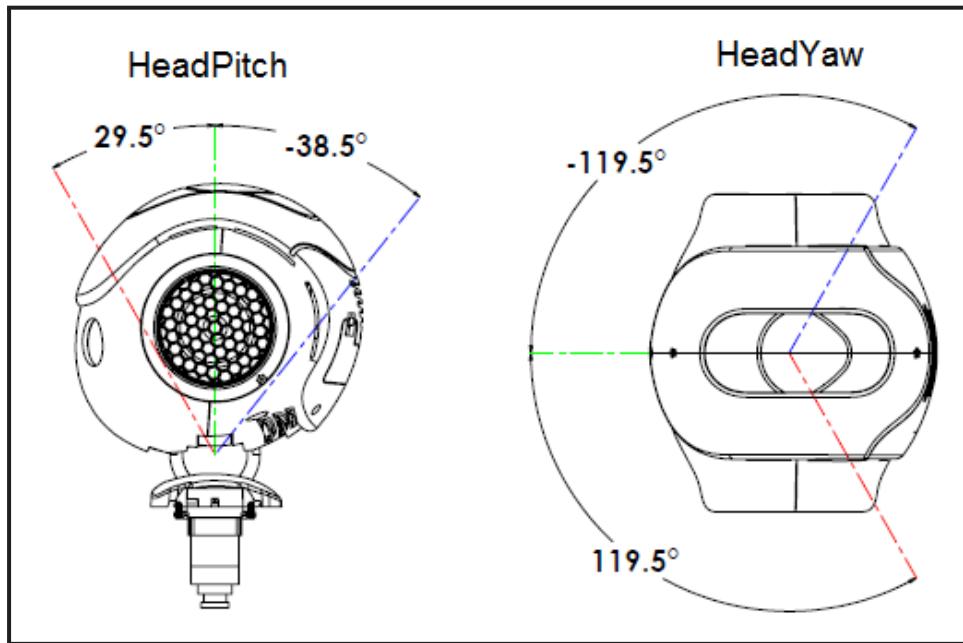
```

Le problème étant que cela ne peut pas fonctionner en temps réel car nous devons fournir un tableau en entrée et que nous ne pouvons pas le modifier. Une solution à ce problème serait d'utiliser un pointeur afin de modifier le tableau directement dans la mémoire, mais cela semblait trop compliqué et donc j'ai décidé de ne pas garder cette solution. J'ai choisie comme solution finale de donner des durées extrêmement petites de mouvement (de l'ordre de 20 millisecondes) exécuté directement sur le Nao. Pour le problème de la vitesse trop élevé (ou appelé aussi vitesse max) il a fallu "simplement" adapter l'angle. Pour une durée donnée, on multiplie par 7.1 ou 8.1 (vitesse max), et si cela est plus grand que la différence d'angle entre l'angle actuel et l'angle futur c'est que cela est trop rapide (rad/s * s = rad). Il suffit ensuite de modifier l'angle de façon à ce qu'il reste dans la limite tout en gardant le mouvement désiré.

Sous tâche 2:

L'étape d'après était d'interpréter les données envoyées par le téléphone en données interprétable par Nao. Les données envoyées par le téléphone étant en degré, il fallait faire une conversion en radians. Il a deux valeurs qui m'intéressent pour le mouvement de la tête.

Le premier est le “Pitch”, c'est à dire le mouvement de haut en bas, et le deuxième le “Yaw”, c'est à dire le mouvement de droite à gauche (voir ci-dessous (doc officiel)):



Sous tâche 3:

Ensuite il a fallu adapter le mouvement. La première étape pour cela est que si l'utilisateur regarde trop à gauche ou trop à droite, le robot doit bouger son corps afin de suivre la vision. Pour cela j'ai dû mettre en place un compteur. Chaque fois qu'un angle donné est trop élevé pour la Yaw, j'incrémente un compteur et s'il dépasse 0.75 secondes le robot tourne sur lui même. Une fois tourner il faut adapter l'angle Yaw qui sera envoyé au robot de façon à ce qu'il prenne en compte le changement. Pour cela il faut faire l'opération suivante:

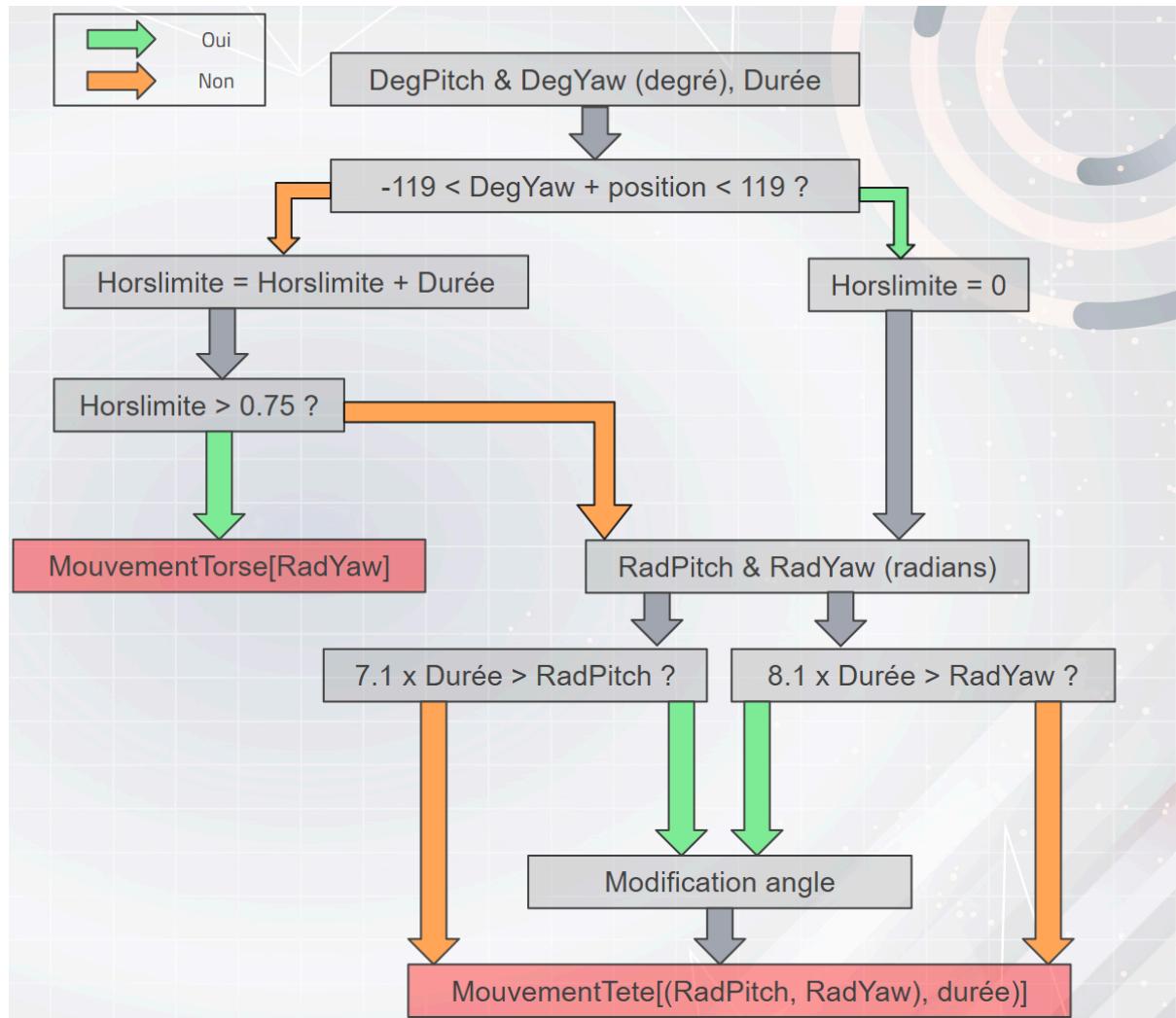
```

● ○ ●

convertAngleYaw(self, angle):
    diff = angle - self.angleYaw
    diff = (diff + 180) % 360 - 180
    return math.radians(diff)
startContinuousSpeechRecognition();

```

Le code au complet vulgarisé donne ceci:



Les difficultés rencontrées:

Les principales difficultés n'ont pas été chaque morceau de code isolé mais de combiner le tout. Aucun de ces codes est compliqué mais gérer en même temps les mouvements de la tête, les exceptions, les mouvements rotatifs du corps, l'angle Yaw par rapport etc ... a été le plus compliqué.

Intégration

Intégration des code du gyroscope, de la géolocalisation et du SpeechToText

Réaliseurs : Remy, Marius, Lazar, Moïra

Rédacteurs : Remy

Description

Les données envoyées au NAO seront les données du gyroscope, celles de localisation et le message du SpeechToText.

On a tout d'abord, les données de gyroscope qui sont le pitch, le roll et le yaw qui représentent les différents axes de rotation avec le temps écoulé en millièmes de secondes pour déterminer le temps d'intervalle entre deux messages. Le roll qui est en axe x, le pitch y et le yaw z.

Ensuite, nous avons le nombre de pas grâce aux capteurs corporels du téléphone. Ce nombre de pas va permettre de faire marcher le NAO d'un certain nombre de pas devant lui.

Et enfin, on a le message retourné par le SpeechToText du téléphone. L'audio du téléphone sera transformé en texte afin de l'envoyer au robot pour qu'il répète ce qu'on dit.

Ces trois données ont été regroupées en un message qui sera envoyé au NAO. Le message est de la forme suivante :

```
Pitch: -0,399814, Roll: 0,287188, Yaw: -134,358780, Time: 2221 Steps: 0, msg:  
chaussettes
```

Intégration des codes Eclipse au projet Android Studio

Réaliseurs : Remy, Marius

Rédacteurs : Remy

Description

Nous avons dû mettre les codes Eclipse qui servaient notamment à envoyer des données au NAO ou recevoir ceux venant de lui sur l'application Android. Dans le cas du flux audio et vidéo du NAO, c'est le téléphone qui est le serveur. Le robot se connecte grâce aux sockets sur le port 9997 pour envoyer les données audios et sur le port 9999 pour envoyer les données vidéos. La vidéo est en 30 images par seconde sur le téléphone. Pour l'audio, du fait que le robot ne peut pas envoyer directement les données en bytes et qu'il faut faire une conversion d'un fichier wav en données en bytes. Ces traitements créent de la latence entre le robot et le téléphone. Pour régler ce problème, Marius a proposé d'accélérer l'audio afin de rattraper le retard que pourrait avoir le téléphone. Cette solution est notamment utilisée par des services de streaming en direct comme Twitch. L'accélération ne se remarque pas et permet de rattraper ces retards qui pourraient arriver à cause de freezes.

Intégration des jambes, tête et mains pour la gestion du déplacement global

Réaliseurs : Léo PHAM

Rédacteur: Léo PHAM

Rectification :

Comme le titre nous l'indique, il s'agit de l'intégration des classes jambes, pieds et mains afin de pouvoir faciliter la gestion du déplacement du NAO.

Sous tâche 1 : Faire une classe commune de déplacement (mouvement)

On a donc importé toutes nos classes, afin de pouvoir créer une classe mouvement qui prend en argument, une adresse IP et ainsi qu'un port afin de pouvoir appeler la classe ALMotion, qui est au cœur pur de nos déplacements. Et voici, à quoi ressemble un peu près notre classe:

```
movement_instance = None
server_running = True
counter = 0
tetes = None

class Movement(object):
    def __init__(self, ip, port):
        global tetes
        self.ip = ip
        self.port = port

        # Les parties des pieds et des mains sont donc activées par ici

        # PARTIE PIEDS
        pieds.left_leg = pieds.LeftLeg(self.ip, self.port)
        pieds.right_leg = pieds.RightLeg(self.ip, self.port)

        # PARTIE MAINS
        bras.right_arm = bras.Arm("RightArm", self.ip, self.port)
        bras.right_arm.initialize_right_arm()
        bras.left_arm = bras.Arm("LeftArm", self.ip, self.port)
        bras.left_arm.initialize_left_arm()
        bras.both_arm = bras.Arm("BothArm", self.ip, self.port)
        bras.both_arm.initialize_left_arm()
        bras.both_arm.initialize_right_arm()

        # PARTIE TETE
        tetes = tete.Head(self.ip, self.port)

    @staticmethod
    def get_right_leg(): return pieds.right_leg
    @staticmethod
    def get_left_leg(): return pieds.left_leg
    @staticmethod
    def get_right_arm(): return bras.right_arm
    @staticmethod
    def get_left_arm(): return bras.left_arm
    @staticmethod
    def get_both_arm(): return bras.both_arm
```

```
import pieds
import tete
import bras
```

En effet, une fois initialisée, elle crée un objet de chaque classe (tête, mains et pieds), on a pu faire un getter pour les pieds et ainsi que les mains puisqu'on gère pas qu'une seule main ni un seul pied... Par ailleurs, c'est pour cela que la tête n'a pas de getter.

Sous tâche 2 : Création de fonctions permettant d'appeler des fonctions dans d'autres classes

Afin de vérifier, tout fonctionne correctement, Léo a tout d'abord, fait un appel assez simple de chaque fonctions dans chaque classe par exemple il a dû faire:

```
pieds.MoveForwardOrBackward(30, "front")
tete.moveHead(-0.39, 0.28, -134.35, 2221)
main_gauche.waving_arm(2)
```

Tout, avait l'air de bien fonctionner, donc on s'était mis à faire des fonctions dans le fichier afin qu'on ait pas à tout retaper ces lignes en continue, par exemple MarcherAuPas, prend en argument pas et puis exécute la fonction du fichier pieds.py, la fonction MoveForwardOrBackward avec comme argument pas et "front". Même principe pour MouvementTete qui prend le pitch, yaw et le temps pour ainsi renvoyer ces arguments pour la fonction de classe head: moveHead.

```
def MarcherAuPas(pas):
    pieds.MoveForwardOrBackward(pas, "front")

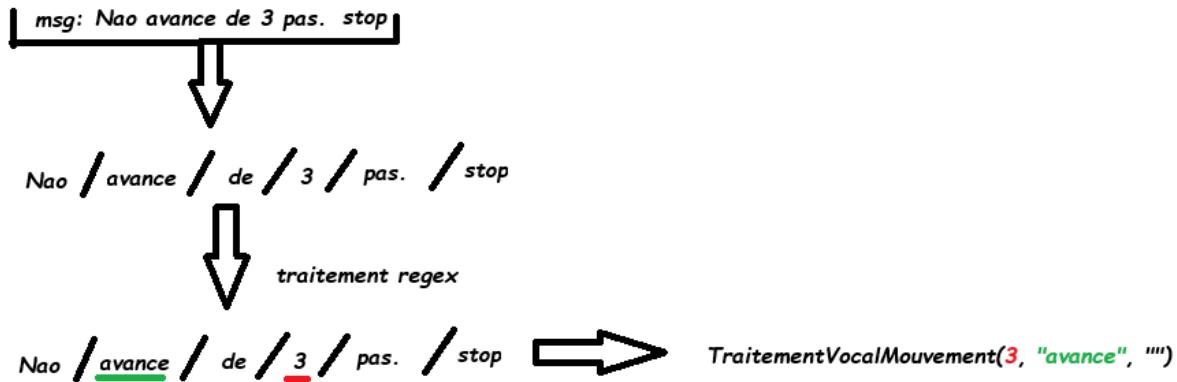
def MouvementTete(pitch, yaw, time):
    global tetes
    tetes.moveHead(pitch, yaw, time)
```

Nous n'avons donc pas fait de fonctions de la sorte, pour les mouvements de mains, car j'avais déjà une idée en tête de le faire via une fonction qui reçoit via un serveur des commandes vocaux... Et puis que Tiancheng a pu faire des fonctions pour chaque mouvement de bras, on s'était donc dit qu'il fallait en profiter puisque dans la sous-tâche suivante, on a pu réaliser ladite fonction.

Sous tâche 3 : Création des fonctions de prétraitement et d'ordre vocaux.

Étant donné qu'il nous était défini dans le cahier des charges, le fait de pouvoir effectuer des commandes vocaux pour le NAO, afin qu'il puisse exécuter ce qu'on lui demande de faire, j'ai donc créé une fonction qui fait du traitement de message, tout d'abord que fait la fonction TraitementVocal, elle prend un message que l'on va recevoir du serveur, pour qu'on puisse ainsi utiliser nos mots prédéfinis (telle que "nao", "agite", "avance", "droite", etc...), ensuite, on transforme notre phrase en une liste de mots, et puis on fait le parcours de la liste, on essaie de comparer nos mots, s'ils correspondent à notre pattern (ici le regex), si c'est le cas, on l'assigne à notre mot d'ordre, notre nombre de pas (ou de fois) et ainsi que le choix des mains.

Pour illustrer tout ça voici un schéma expliquant, tout ce qu'on venait de décrire :



(Évidemment faut que le message contient le nao afin de pouvoir faire ceci), sinon il le dira à voix haute voix (implémentation d'une des classes de Keryann)

Voici ce que cela nous donne en code

```

def TraitementVocal(message):
    # Initialisation des variables
    nb_fois = None
    type_mouv = None
    choix_main = None
    HasNaoInIt = False
    keyword = re.compile(r"(nao)")

    # On assigne tous les patterns pour chaque commande vocal
    nb_fois_pattern = re.compile(r"(\d+)\s*fois")
    type_mouv_pattern = re.compile(r"(salue|agit|celebre|leve|baisse|avance|recule)")
    choix_main_pattern = re.compile(r"(droite|gauche|tous)")

    words = message.split() # On sépare tous les mots du message dans une liste de mots

    for word in words:
        # On regarde si les mots correspondent aux patterns
        nb_fois_match = nb_fois_pattern.match(word)
        type_mouv_match = type_mouv_pattern.match(word)
        choix_main_match = choix_main_pattern.match(word)
        keyword_match = keyword.match(word)

        if word.match(word):
            # On assigne tous les mots (ou groupe de mots) aux variables pour l'exécution des mouvements via la
            # fonction TraitementVocal
            if nb_fois_match:
                nb_fois = int(nb_fois_match.group(1))
            elif type_mouv_match:
                type_mouv = type_mouv_match.group(1)
            elif choix_main_match:
                choix_main = choix_main_match.group(1)
            if keyword_match:
                HasNaoInIt = True

    if HasNaoInIt:
        TraitementVocalMouvement(nb_fois, type_mouv, choix_main)
    else:
        DireMessage(message)
  
```

Et pour ce qui de la fonction TraitementVocalMouvement:

```

def TraitementVocalMouvement(nb_fois, type_mouv, choix_main):
    bras_gauche = movement_instance.get_left_arm()
    bras_droit = movement_instance.get_right_arm()
    les_deux_mains = movement_instance.get_both_arm()

    if type_mouv.lower() == "salue":
        if choix_main.lower() == "droite":
            bras_droit.wavearm(nb_fois)
        elif choix_main.lower() == "gauche":
            bras_gauche.wavearm(nb_fois)
        elif choix_main.lower() == "tous":
            les_deux_mains.waving_both_arm(nb_fois)

    elif type_mouv.lower() == "agite":
        if choix_main.lower() == "droite":
            bras_droit.agiter_arm()
        elif choix_main.lower() == "gauche":
            bras_gauche.agiter_arm()
        elif choix_main.lower() == "tous":
            les_deux_mains.agiter_arm()

    elif type_mouv.lower() == "celebre":
        les_deux_mains.celebrate()

    elif type_mouv.lower() == "leve":
        if choix_main.lower() == "droite":
            pieds.LiftLeg(30, "right")
        elif choix_main.lower() == "gauche":
            pieds.LiftLeg(30, "left")
        else:
            bras_droit.dontUnderstand_arm(5)

    elif type_mouv.lower() == "baisse":
        if choix_main.lower() == "droite":
            pieds.LiftLeg(30, "right")
        elif choix_main.lower() == "gauche":
            pieds.LiftLeg(30, "left")
        else:
            bras_droit.dontUnderstand_arm(5)

    elif type_mouv.lower() == "avance":
        pieds.MoveForwardOrBackward(nb_fois, "front")

    elif type_mouv.lower() == "recule":
        pieds.MoveForwardOrBackward(nb_fois, "back")

    else:
        bras_droit.dontUnderstand_arm(5)

```

Selon l'argument en entrée pour, le NAO ira effectuer un de mouvements suivants, si par exemple les arguments sont nb_fois = 3, type_mouv = “avance” et choix_main = “”, il ira donc faire l'action pieds.MoveForwardOrBackward, 3 fois (3 pas du coup) en avançant. Si jamais il ne comprends, pas il n'a plus qu'à lancer l'animation “dontUnderstand_arm”

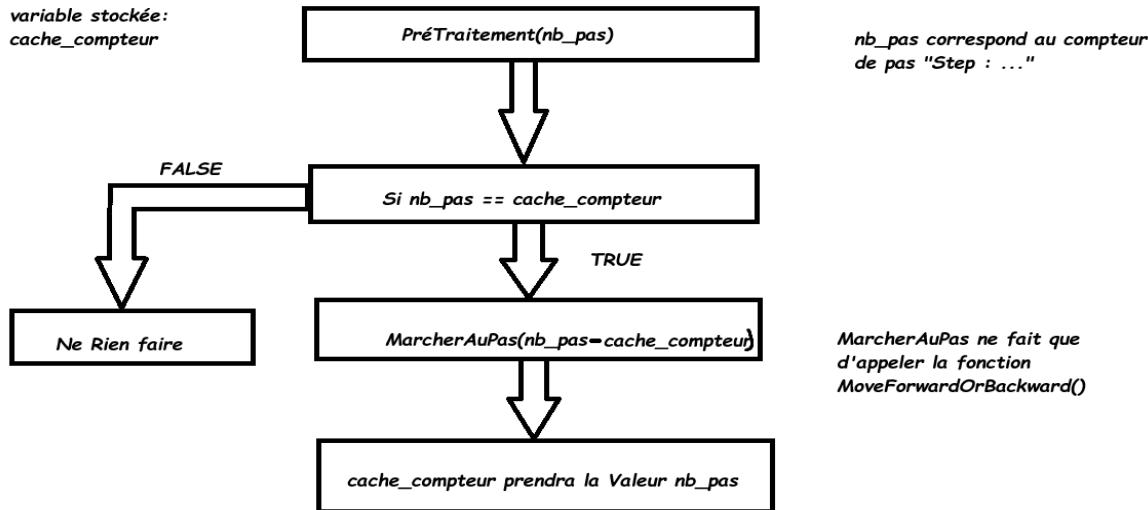
Sous tâche 4 : Faire avancer le NAO avec compteur de pas !

Étant donné que Lazar ait fait un compteur de pas, et qu'il est envoyé sur le serveur et que le serveur se rafraîchit toutes les 0.2 secondes, il faudrait qu'on fasse une fonction qui permet de comparer, le nombre de pas d'avant et le nombre de pas actuel (sous risque de le faire avancer infiniment). On a donc fait des variables en “mémoire cache” et donc fait une fonction qui permettait de comparer, si le nombre de pas étant différent du nombre de pas stocké en cache, alors on fait une simple soustraction entre le nombre actuel et le nombre stocké en cache et on lui fait avancer la différence en pas. Enfin, la variable en cache va donc stocker la variable actuelle.

```

# Variables de cache
cache_msg = ""
movement_instance = None
server_running = True
counter = 0
tetes = None
tts = None

```



Par ailleurs, on peut aussi constater que c'est le même principe pour le prétraitement des messages, sauf qu'il n'y a besoin d'effectuer une soustraction.

Intégration dans le serveurs python du traitement des mouvements

Réaliseurs : Marius, Gabriel, Léo
Rédacteur: Gabriel

Sous tâche ?:

En se basant sur le code produit afin de communiquer entre Nao et le téléphone, il a été possible de faire les mouvements de la tête en temps réel. Les données envoyées par le téléphone sont sous forme de String (liste de caractères, une phrase). Étant un flux en continu, il n'y a pas de pause de un nouveau String est envoyé en continu. Cela fait que seul des morceaux de chaque String est envoyé et jamais un seul entier. Il a donc fallu mettre en place une solution pour séparer les Strings entre eux. La solution retenue a été de mettre un caractère distinct à la fin de chaque String envoyé (un "\n" signifiant fin de ligne). Côté Nao il a donc fallu faire ce qu'on appelle un "buffer", un "tampon" en langue de Molière. Ce buffer va additionner tous les Strings reçus pour n'en faire qu'un, et à chaque réception d'un nouveau String il va regarder en additionnant celui ci avec ce qu'il a déjà en stock, s'il y a un "\n" et donc un String de complet. Si oui il va envoyer le String au reste du code qui va le traiter. Un problème qui a été trouvé à ce moment est que si le robot tourne, il ne peut rien faire entre-temps et met donc le code en pause. Mais une fois sa pause finis (~10 secondes), il reprend les instructions là où elles étaient et va donc avoir un retard de 10 secondes sur le mouvement du téléphone. La solution a été de dire de créer une horloge au lancement du serveur, et lors de chaque traitement de String, le code regardera si le temps fournie dans le string est supérieur au temps qu'il a localement avec une marge de 50 millisecondes. L'inconvénient est donc que l'on ne peut pas avoir plus de 50 millisecondes (ou un autre temps si celui ci est modifié) de latence.

Les difficultés rencontrées:

La principale difficulté a été d'identifier le fait que les Strings sont incomplets à leur réception. L'autre difficulté a été de garder une synchronisation entre le téléphone et Nao.

Conclusion

Afin de conclure ce projet, nous allons lister les points positifs et les points négatifs.

Tout d'abord, dans un premier temps, nous avons surmonté toutes les difficultés majeures, par exemple l'implémentation d'un compteur de pas dans le téléphone qui a pour but de recréer un mouvement relatif ou encore l'accélération du flux audio joué sur le téléphone pour réduire la latence. Des solutions ont été trouvées à toutes les difficultés majeures rencontrées. Cependant le projet n'a pas pu être finalisé dans les délais impartis, une rallonge d'une semaine nous aurait permis de finaliser et parfaire le code et la correction de certains bugs.

De plus, une forte cohésion de groupe a été une vraie plus-value dans la réalisation de ce projet. Des amitiés et des bons moments se sont créées. Malgré cela, nous avons eu affaire à de légers problèmes de communication qui ont entraîné des retards dans la réalisation.

Il est à noter que certaines consignes ont été contradictoires. Les modalités de rendu et d'évaluation ont été modifiées tardivement à plusieurs reprises tout au long projet, ce qui nous a menés en erreur et a retardé le démarrage du rendu final. A contrario, une liberté nous a été donnée sur la réalisation du projet ainsi que sur sa présentation. Cette liberté nous a permis d'interpréter le projet à notre manière et d'innover dans les solutions apportées.

Enfin, un manque cruel de documentation couplé à une absence globale de connaissance des robots NAO à l'IUT se sont fait ressentir. En parallèle, des problèmes techniques des Nao pour la plupart, indépendants de notre volonté ont émergé. L'addition des deux a entraîné un fort retard en début de projet. En contrepartie, ces problèmes nous ont permis d'utiliser nos compétences techniques apprises à l'IUT comme les connexions réseaux et les threads vus en S3 et le développement mobile appris en S4. Durant ce projet, nous avons dû faire preuve d'adaptation pour répondre aux différentes attentes.

Ce projet a été complexe mais stimulant. Il nous a permis d'innover, d'apprendre de nouvelles choses et d'expérimenter un projet avec une équipe conséquente. Ce projet a eu son lot de problèmes que nous avons surmonté.