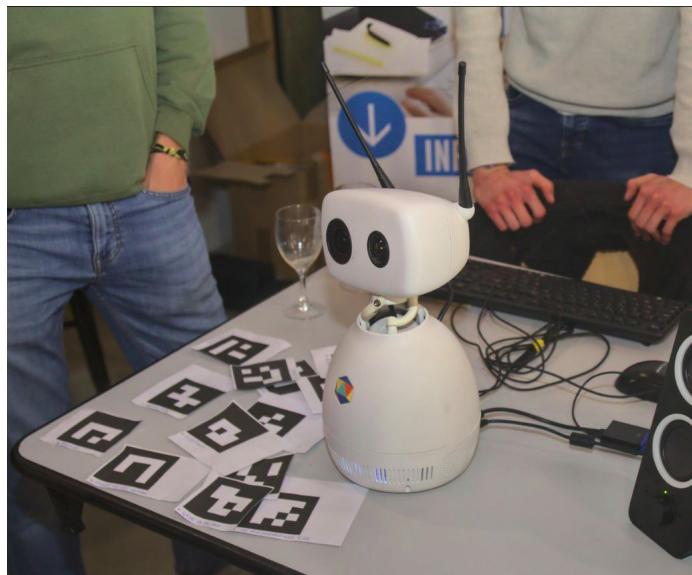


FILIÈRE INFORMATIQUE - SEMESTRE 8

RAPPORT

Application d'assistant vocal sur Reachy Mini



Auteurs :

BOUDEAU Benjamin
DIEUDONNÉ Clara
ELFANI Hamza
LAMHAMDI Aymane
MARAIS Lucas
RAÏS Sylvain
ZIZOUAN Widad

Encadrant :
ROLLET Antoine
MORANDAT Floréal
Client :
N'GUYEN Steve

Table des matières

1 Introduction	3
2 Organisation et méthodes agiles	4
2.1 Utilisation de ClickUp	4
2.2 Répartition des tâches	4
2.3 Utilisation de Github	6
2.4 Organisation des sprints et livrables	6
2.5 Les tests réalisés	7
3 Architecture de notre projet	7
3.1 Organisation de notre code	7
3.2 Utilisation d'une session	8
3.2.1 Première implémentation	8
3.2.2 Soucis engendrés	8
3.2.3 Solution envisagée	9
3.3 Utilisation d'une machine à états	9
3.3.1 Première implémentation	9
3.3.2 Implémentation de la machine à états généralisée	9
3.3.3 Notion d'exécuteur	10
3.3.4 Fichiers JSON	11
3.3.5 Notion de Timeout pour un état	13
3.3.6 Lancer une machine à états sur le Reachy	13
4 Les différentes fonctionnalités	14
4.1 Reconnaissance vocale	14
4.1.1 Gestion des mots-clé	14
4.1.2 Gestion prise du son	15
4.1.3 Réduction du bruit	15
4.1.4 Conversation avancée	15
4.2 Synthèse vocale	16
4.2.1 Utilisation de pyttsx3	16
4.2.2 Utilisation de gTTS	17
4.3 Traitement d'image	18
4.3.1 Prise de photos	18
4.3.2 Filtres	21
4.3.3 Gestion des photos : Enregistrement, affichage et suppression	23
4.4 Détection des tags Arucos	24
4.5 Mouvements	25
4.5.1 Faire différentes émotions	25
4.5.2 Suivi et cadrage des utilisateurs	26
4.5.3 Ressemblance aux mouvements humain	27
5 Mise en place et utilisation du robot / Manuel d'utilisation	28
5.1 Mise en place et matériel nécessaire	28
5.2 Initialisation du robot	29
5.3 Utilisation du robot	30
5.3.1 Machine à états quelconque	30
5.3.2 Machine à états "Only Aruco"	31
5.3.3 Machine à états "finale"	31
5.4 Autres programmes utiles	31
5.4.1 Programme d'affichage des photos	31
5.4.2 Programme de suppression de photos	31
6 Les limites du robot et les améliorations possibles du projet	31
6.1 Les qualités et défauts de Reachy Mini	32
6.2 Améliorations possibles de notre projet	32
7 Évènements de présentation du projet	33
7.1 Soirée Partenaires	33
7.2 100 ans de l'ENSEIRB	33

8 Conclusion	34
9 Annexes	36
9.1 Cahier des charges	36
9.2 Fichier .json	62

1 Introduction

Le Projet au Fil de l'Année, aussi appelé PFA, consiste en la réalisation d'un projet concret proposé par un client à 7 élèves ingénieurs de l'ENSEIRB-MATMECA. Ce rapport a pour but de détailler la réalisation du projet concernant le robot Reachy Mini de l'entreprise Pollen Robotics. Ce projet a été proposé par N'GUYEN Steve, client membre de l'entreprise ayant développé le robot.

Reachy Mini

Le robot Reachy Mini est un robot articulé. Cependant il se distingue des robots habituels et notamment de son grand frère, le robot Reachy, par son unique buste dépourvu de jambes. Ce sont sa tête, et ses antennes qui sont articulées à 360°, dans la limite des câbles électriques passant par le coup et permettant l'alimentation de la caméra située dans la tête. En effet, le robot possède deux caméras à la place de ses yeux, il peut donc voir et capter l'environnement autour de lui. Monté sur un PC au processeur Intel NUC et un Google Coral, embarquant un micro multi-directionnel et deux haut-parleurs, le robot possède les outils nécessaires à l'analyse de l'environnement et l'interaction avec le monde qui l'entoure.

Pollen Robotics

Fondée en 2016 par d'anciens chercheurs, Pollen Robotics est une entreprise qui rassemble des personnes talentueuses et indépendantes dans le but de fournir des produits et des applications accessibles et open source. L'entreprise participe donc à l'évolution de l'IA et de la robotique et est motrice à son intégration dans la vie quotidienne. Elle expose ainsi son robot principal Reachy sur de nombreux événements autour de l'informatique, la robotique et l'intelligence artificielle.



FIGURE 1 – Logo de Pollen Robotics

Le projet

Le travail demandé par le client pour réaliser ce projet était de programmer Reachy afin qu'il se rapproche d'un assistant vocal du type Amazon Echo, ou Google Home. La reconnaissance vocale, le traitement d'images, la synthèse vocale et les mouvements du robot étaient donc à implémenter.

L'objectif principal du projet était la présentation du robot lors des 100 ans de l'ENSEIRB-MATMECA, le 8 avril 2022. Cependant, l'événement a été déplacé au 30 septembre. L'objectif principal a donc été la présentation du robot lors de la soirée des partenaires de l'école, événement qui sera détaillé dans ce rapport. Le robot devait alors être capable de prendre des photos, et interagir avec les utilisateurs.

D'autres fonctionnalités demandées par le client consistaient à pouvoir demander les conditions de surf en un lieu et une journée précise. Le robot devait ainsi aller sur internet et effectuer cette recherche avant d'informer l'utilisateur avec la réponse.

Ce rapport se décompose en plusieurs parties, la première présentera l'organisation du projet tout en respectant les méthodes agiles. Ensuite, nous présenterons l'architecture de l'implémentation de notre projet avant de présenter les différentes attentes du projet et de les comparer à la réalisation effectuée. Enfin, nous présenterons le manuel d'utilisation du robot et ses limites avant de finir par présenter l'événement de la soirée partenaires.

2 Organisation et méthodes agiles

Afin d'organiser la réalisation du projet, il a été nécessaire de respecter les méthodes agiles. Ce projet a ainsi été organisé autour de sprints permettant le développement de solutions et la consultation avec le client pour vérifier le bon respect des contraintes et besoins.

Afin de coordonner le travail entre les 7 membres du groupe, un gestionnaire de tâches a été utilisé. Pour réaliser ce projet, c'est ClickUp qui a été choisi comme gestionnaire.

2.1 Utilisation de ClickUp

ClickUp permet de centraliser les fonctions collaboratives d'un projet. L'outil regroupe en effet de nombreuses fonctionnalités et permet ainsi de n'utiliser qu'une unique interface de gestion au lieu de multiplier les outils. ClickUp facilite donc la coordination au sein du groupe. L'outil permet de créer des listes de tâches simples et des listes plus complexes avec des sous-tâches, attribution de rôle, ... Il permet également la génération de calendriers correspondant à ces tâches et leurs dates de réalisation ou deadline. Les différents diagrammes de Gantt peuvent ainsi être générés directement sur ClickUp en liant les tâches entre elles avec des dépendances.

Certains outils comme le tableau blanc, ou les notes sont aussi utiles pour la coordination du groupe, notamment avec les réunions à distances qui sont facilitées par cet outil.

Ainsi, nous avons choisi d'utiliser ClickUp pour son interface intuitive, sa modernité et ses nombreuses fonctionnalités permettant de centraliser une grande partie de la gestion du projet sur un seul outil.



FIGURE 2 – Logo ClickUp

2.2 Répartition des tâches

En fonction des besoins fonctionnels et non fonctionnels identifiés entre l'équipe et le client au début du projet, différentes équipes ont été formées pour pouvoir avancer le projet en parallèle. Puis chaque implémentation interdépendantes ont été regroupées pour permettre le fonctionnement global du robot.

L'un des premiers besoins fonctionnels était les mouvements du robot. En effet, lors de son interaction avec les utilisateurs, ce dernier devait être possible de réaliser des mouvements afin de notamment regarder l'interlocuteur, mais aussi faire des émotions telles qu'être content, ou triste.

Ainsi, Lucas et Clara ont formé la première sous-équipe et se sont concentrés sur les parties liées au mouvement du robot. L'utilisation de l'API `reachy-sdk` a permis d'utiliser les fonctions bas niveau de Reachy pour implémenter tous les mouvements réalisables par le robot.

Ensuite, le robot devait être en mesure de parler afin d'interagir avec un potentiel utilisateur. Aymane et Hamza ont donc étudié la synthèse vocale en cherchant à donner au robot une voix cohérente et en créant un système de parole.

Pour pouvoir parler aux utilisateurs, il était nécessaire que le robot comprenne ce qu'ils pouvaient lui dire. La reconnaissance vocale a donc constitué une troisième partie du projet. Benjamin et Widad ont ainsi travaillé sur la compréhension d'un texte oral et son analyse par le robot. Ensuite Benjamin s'est tourné vers la mise en place de machines à états plus complexes et Widad sur la mise en place des conversations avancées.

Enfin, comme l'un des objectifs était la prise de photo par le robot, Sylvain a constitué la dernière sous-équipe en travaillant avec l'API `OpenCV` pour analyser l'image captée par les caméras du robot et implémenter

la prise de photos.

Quatre équipes distinctes ont ainsi été constituées. Certains sous-objectifs du projet tels que la mise en place de la machine à états permettant le fonctionnement du robot ou encore la mise en place des actions parallèles, ont entraîné la création de nouvelles sous-équipes avec des tâches en parallèle de leurs tâches principales.

ClickUp a permis de centraliser les étapes de la réalisation de ces différentes tâches. Le maintien régulier de l'avancée de ces tâches permettait donc de connaître le retard pris sur certaines étapes ou l'avancée prise sur d'autres, et donc de corriger en apportant plus d'aide sur une étape plutôt que sur une autre.

Ainsi, le diagramme de Gantt initial a évolué au cours du projet. En effet, ces retards ou avancées ont eu des impacts légers sur la planification du projet. Également, certaines fonctionnalités n'avaient pas été réfléchies de la même façon qu'elles ont été finalement implémentées. Ces légers changements ont donc eu un impact important dans la planification finale du projet, remettant en cause certaines parties du projet ou leur implémentation.

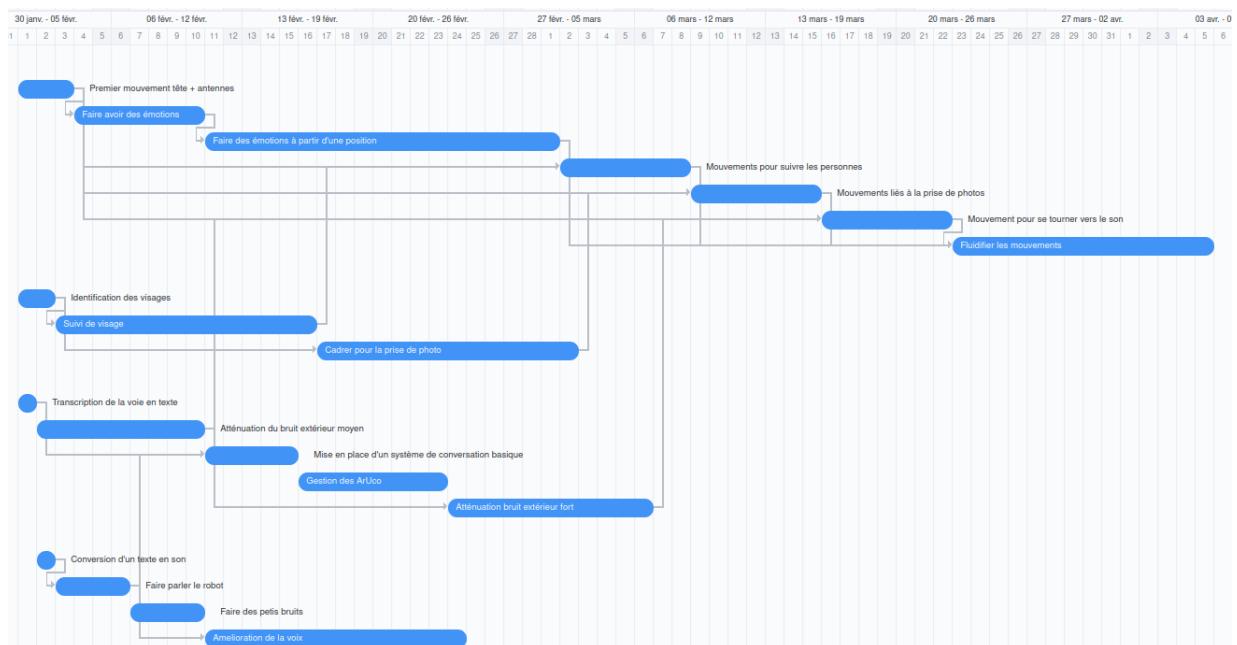


FIGURE 3 – GANTT initial avant les 100 ans

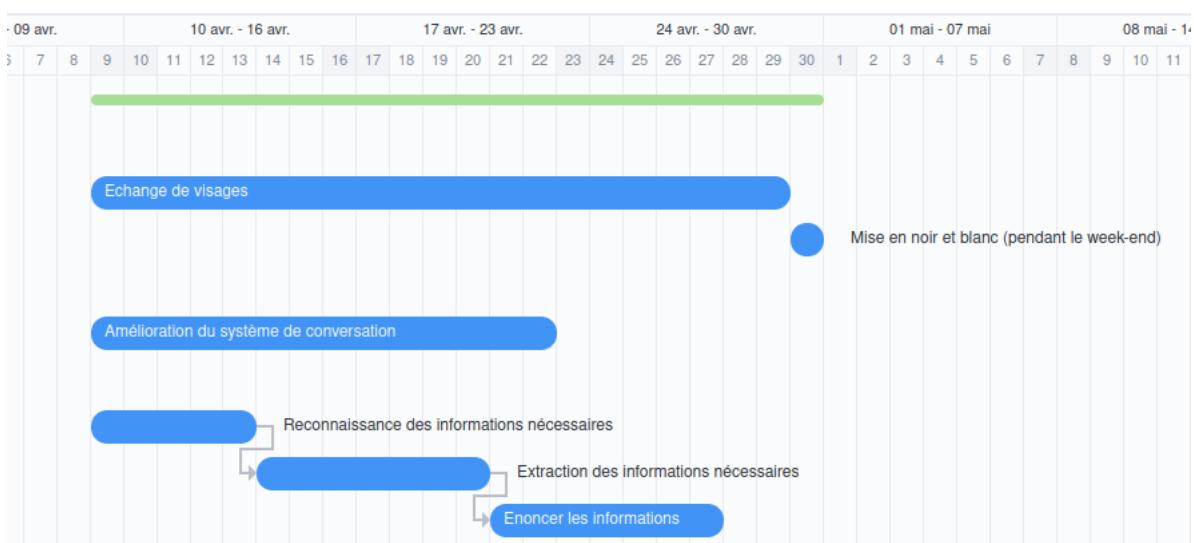


FIGURE 4 – GANTT initial après les 100 ans

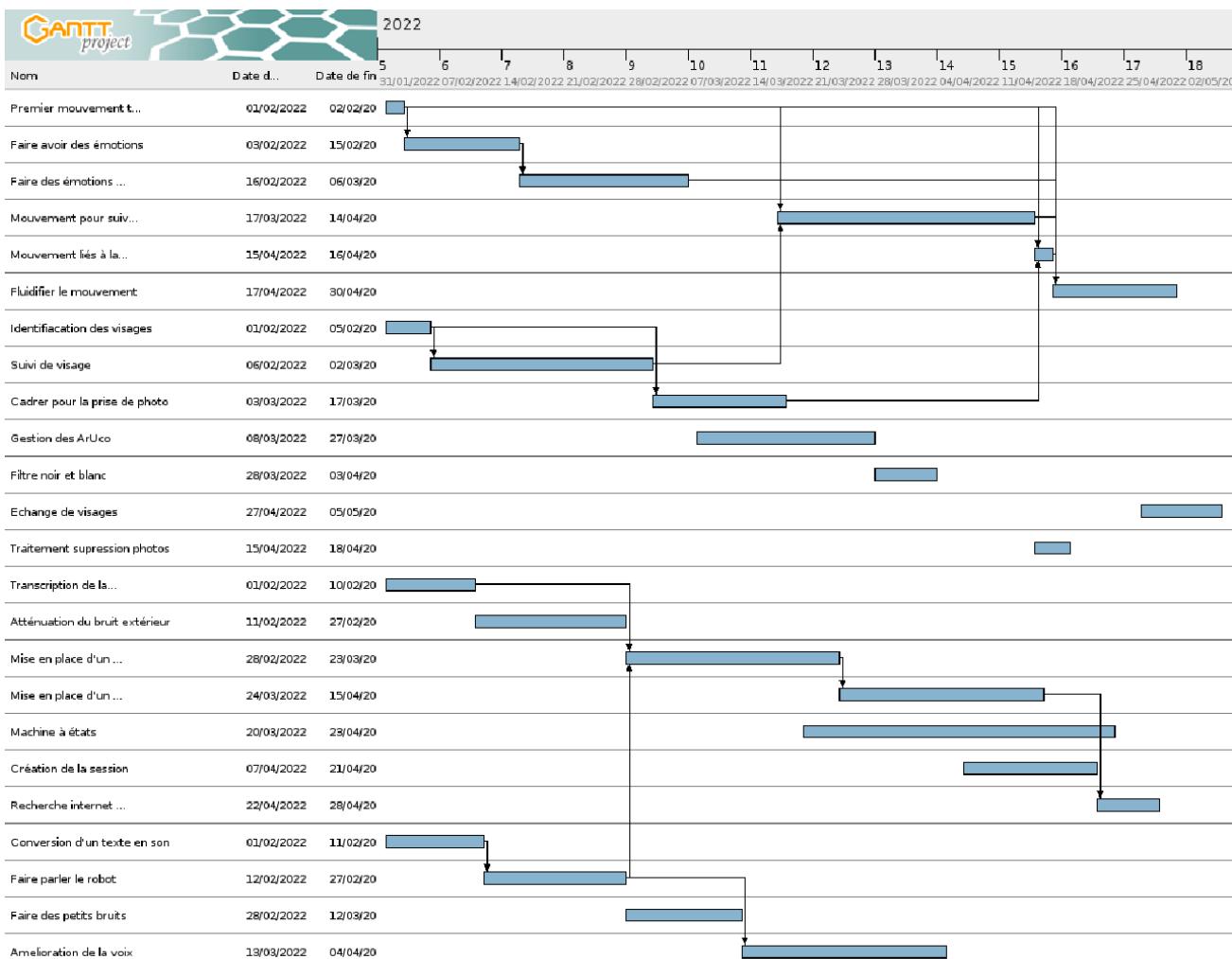


FIGURE 5 – GANTT final

2.3 Utilisation de Github

Pour permettre le partage du code au sein de l'équipe mais également avec l'encadrant, plusieurs solutions étaient possibles.

La première consistait à utiliser les serveurs de l'école et héberger le projet sur un dépôt git lié à la forge de l'école. Cependant, suite à quelques latences fréquentes ayant eu lieu en début d'années sur Thor, nous avons choisi de ne pas utiliser cette méthode.

Ensuite, nous avions le choix entre plusieurs gestionnaires de dépôt git tels que Github ou Gitlab. C'est avec unanimité que nous avons choisi Github, étant tous déjà utilisateurs de cet outil et familiers avec son utilisation.

Le robot a d'ailleurs été connecté au compte Github de Lucas. De nombreux commits ayant été réalisés depuis le robot, ce compte possède une grande partie du code du projet, bien que toute l'équipe ait travaillé et codé avec le robot.

2.4 Organisation des sprints et livrables

Le management du projet a été organisé selon les principes des méthodes agiles. Pour cela, différents sprints ont été réalisés. Un sprint consistait en la mise en commun d'objectifs avec l'encadrant et/ou le client, puis la réalisation de ces objectifs pendant 2 semaines avant une deuxième réunion de validation de la réalisation. Différents livrables ont ainsi pu être présentés respectant les étapes de validation de ces sprints.

Le rythme moyen de réunions consistait à réaliser une réunion par semaine soit avec l'encadrant, soit avec le client. Cependant, nous avons eu un deuxième encadrant (Mr MORANDAT) au milieu du projet, ce qui a légèrement impacté ce rythme régulier de réunion. En effet, ce nouveau regard extérieur sur le projet, a permis

de lever de nouvelles idées notamment sur la gestion des mouvements du robot et de la machine à états. Il a donc fallu revoir certaines notions de l'implémentation et donc mettre en stand-by la réalisation du livrable suivant pour le client.

Nous avons, par ces changements, permettant une implémentation plus rapide, pu reprendre le rythme normal des sprints par la suite.

Afin de développer les livrables, nous avons utilisé différentes branches du dépôt git, pouvant ainsi conserver un livrable fonctionnel et continuer de développer la suite du projet en parallèle. De plus, ces branches ont permis de tester différentes solutions à différents problèmes sans impacter la globalité du projet. La bonne intégrité de la branche principale (**master**), est en effet un des aspects caractéristiques des méthodes agiles. Au maximum, les commit et pushes étaient donc réalisés sur une branche parallèle puis intégrée au reste du code après validation.

2.5 Les tests réalisés

Afin de réaliser des tests sur l'implémentation du projet, nous avons utilisé différentes méthodes.

La première consistait en tests de recette. En effet, la plupart des fonctions du robot étant des fonctions de mouvement utilisant l'API, elles ne retournent pas forcément de valeurs. Il a donc été visuellement possible de tester certaines parties du code en appelant certaines fonctions et en observant leur bonne exécution.

Par exemple, tout le code lié aux photos ne retourne pas de valeurs mais manipule des fonctions. Pour les tester, nous avons donc vérifié la bonne réalisation des attentes par le robot.

Nous avons cependant réalisé quelques tests unitaires sur certaines fonctions comme `inverse_kinematic` par exemple en vérifiant sa valeur de retour qui parfois est `null` car elle n'a pas trouvé de solution au quaternion.

En ce qui concerne la synthèse vocale ou la reconnaissance vocale, il n'était possible de réaliser que des tests visuels, tests de recette. En effet, il n'est pas possible de tester si le robot capte bien le son autrement qu'en parlant et regardant ce qu'il détecte. Pareil pour la synthèse où il est impossible de tester autrement qu'en faisant parler le robot et vérifiant qu'il a dit la bonne chose.

Ainsi, ce projet se portait peu à l'utilisation de test unitaire ou d'intégrité. Cependant les tests de recettes ont été très utilisés.

3 Architecture de notre projet

Un aspect important du projet est aussi l'organisation et l'architecture de notre code afin de le rendre plus fonctionnel et modulable.

3.1 Organisation de notre code

L'implémentation de l'ensemble du code du projet constituant un nombre important de lignes de code, nous avons donc séparé le code du projet en plusieurs fichiers sources. Chaque fichier correspondant à une partie du projet (Mouvement, Traitement d'image, ...). En plus des fichiers qui codent l'implémentation du projet, il y a aussi des fichiers permettant la description et l'exécution du projet.

Nous avons donc créé différents dossiers pour organiser tout cela. Dans un premier temps, nous avons fait une séparation entre les fichiers qui sont utilisés par le code : `assets`, le code en lui-même : `src`, les fichiers créés par le code : `tmp` et les tests : `tst`.

Le dossier `assets` contient l'ensemble des commandes vocales pour passer d'un état à un autre, les fichiers `json`, pour créer la machine à états dont le fonctionnement sera détaillé dans la partie 3.3, les sons que Reachy émet.

Le dossier `src` est séparé en 6 parties, un dossier contenant le code pour la détection d'image `detection`, un pour la session, détaillé en 3.2, un pour la synthèse vocale `speech`, un pour les machines à états `state_machine`, et un pour la gestion mémoires pour l'enregistrement des photos `storage_management`. Enfin, certains fichiers ne faisant pas partie d'un groupe de fichiers, sont simplement dans le répertoire `src` comme le mouvement ou

la reconnaissance vocale.

Le dossier `tmp` permet de stocker les images et les sons enregistrés par le robot.

Le dossier `tst` contient les fichiers permettant de vérifier le fonctionnement de notre code.

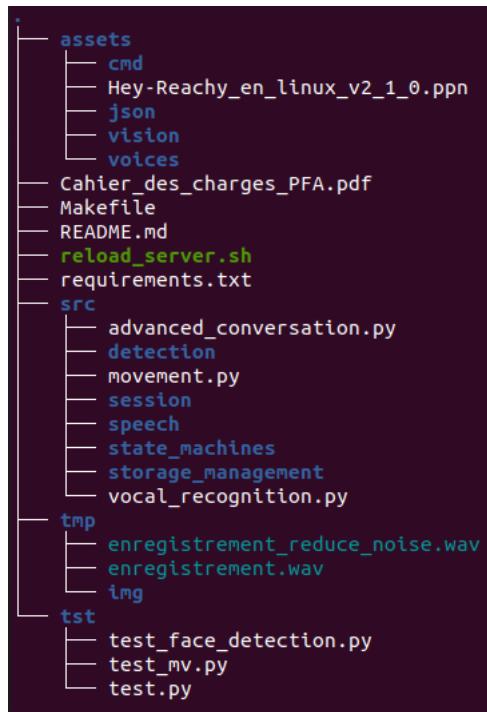


FIGURE 6 – Arborescence de notre code

3.2 Utilisation d'une session

Pour permettre l'abstraction des fonctions bas niveau de l'api `reachy-sdk`, nous avons utilisé le principe de session.

3.2.1 Première implémentation

Afin de gérer les mouvements du robot, nous avions initialement créé un simple fichier `movement.py` qui contenait les fonctions des mouvements telles que celles des émotions (`sad`, `happy`, `listen`, ...), celles permettant le mouvement du robot pour suivre un visage, et celles permettant le mouvement des antennes.

Ce fichier constituait ainsi une classe `Movement` qui avait alors un `ReachySDK` comme attribut, et la connexion au robot était réalisée par l'intermédiaire de cet attribut. Cependant, l'usage de la caméra nécessitait également une connexion au robot. Il fallait donc se connecter au robot de deux façons différentes dans le main de la machine à états afin de créer ces liens.

3.2.2 Soucis engendrés

Avec cette implémentation, l'utilisation du robot nécessitait alors la connexion à ce dernier de deux façons. Nous avions donc des problèmes de liens avec le robot. En effet, la plupart du temps, il y avait deux instances de créées et non une unique qui servait pour le mouvement et la caméra.

Également, avec cette méthode, la connexion au robot étant réalisée dans le main, il y avait du code gérant la connexion dans la machine à états, alors qu'il n'y a pas de liens directs entre les deux modules. Ainsi, il fallait trouver un moyen de ne réaliser qu'une unique connexion et hors du fichier de la machine à états.

De plus, afin de réaliser les tests, nous avions besoin d'un faussaire, c'est-à-dire un faux robot, permettant de tester nos fonctions sans nécessité de connexion au réel robot. Avec cette implémentation, il était donc nécessaire de créer deux fichiers `movement.py` et `mock_movement.py`. Or cela impliquait de la duplication de code.

3.2.3 Solution envisagée

La solution envisagée a consisté en la création d'une session. Une session est un objet manipulé par les états et transitions de la machine à états et permettant la connexion au robot depuis la création du contexte de cette machine à états.

La session est donc une classe abstraite des fonctions bas niveau du robot utilisée dans l'intégralité du projet (mouvement, caméra, ...). Ensuite, nous avons pu développer deux sous-classes qui étendent cette classe abstraite : `ReachySession` et `FakeSession`. Ainsi, la classe correspondant à la session d'un réel robot étend les fonctions bas niveau en appelant celles du robot alors que la session faussaire, ne réalise que des affichages permettant de voir quelles fonctions ont été appelées.

Le fichier permettant les mouvements implémente ainsi les fonctions des mouvements en passant en paramètre une session. Ainsi, chaque mouvement est réalisé sur une session, et c'est en fonction de la session passée en paramètre que ces mouvements sont réalisés sur le robot ou sur le faussaire.

Les mouvements ne sont donc plus une classe mais un simple fichier de définition des méthodes permettant au robot de bouger.

Ainsi, depuis la machine à états, on peut appeler les fonctions de mouvement avec comme paramètre le champ `session` du contexte de la machine à états.

3.3 Utilisation d'une machine à états

Dès le début du PFA nous avons choisi d'utiliser une machine à états afin de spécifier les fonctionnalités du robot dans le cahier des charges. Nous avons choisi cette manière de spécifier car celle-ci se prête bien pour un robot, le Reachy peut s'apparenter à un ensemble d'états et de transitions avec des prédictats qui permettent de cheminer correctement dans cet ensemble d'états. Lorsque nous avons commencé à coder, l'une des premières choses que nous avons faite a été d'implémenter cette machine à états. Cela s'est fait en plusieurs temps que nous allons expliquer maintenant.

3.3.1 Première implémentation

La première implémentation de la machine à états se basait sur un système de dictionnaires. Il y avait un dictionnaire avec comme clé les noms des états et comme valeur la fonction associée à l'état. Chaque fonction d'état commençait par effectuer l'action de l'état puis ensuite il y avait une phase de détection de transition implémentée par une succession d'instructions conditionnelles.

Cette méthode a vite montré ses limites, lorsqu'il s'agissait de rajouter une transition, la fonction de l'état associé devenait rapidement illisible. De plus, il était compliqué de créer d'autres machines à états sans devoir tout refaire. C'est pourquoi, avec les conseils de M. Morandat, nous nous sommes lancés dans l'implémentation d'un objet `State_Machine` dont le but était de généraliser cette notion de machine à états et de pouvoir ainsi créer des sous machines de tests qui font par exemple exclusivement la photo ou bien la conversation.

3.3.2 Implémentation de la machine à états généralisée

Afin de généraliser la notion de machine à états il a fallu se poser la question des éléments qui la composent. Il y a donc les états et les transitions, ces éléments sont eux-mêmes découpés en sous-éléments. Pour un état nous trouvons son action et son ensemble de transitions sortantes. Pour une transition il y a un état de sortie, un prédictat ainsi qu'une action. Lors des débuts de la généralisation de l'objet `State_Machine`, nous avons créé un objet pour les états et un autre pour les transitions. Finalement nous avons trouvé cela plus simple de rester sur seulement un objet `State_Machine` et de représenter les états et les transitions de manières différentes.

Afin d'expliquer tout cela plus simplement, nous allons nous baser sur la Figure 7 qui résume la structure de la `State_Machine`.

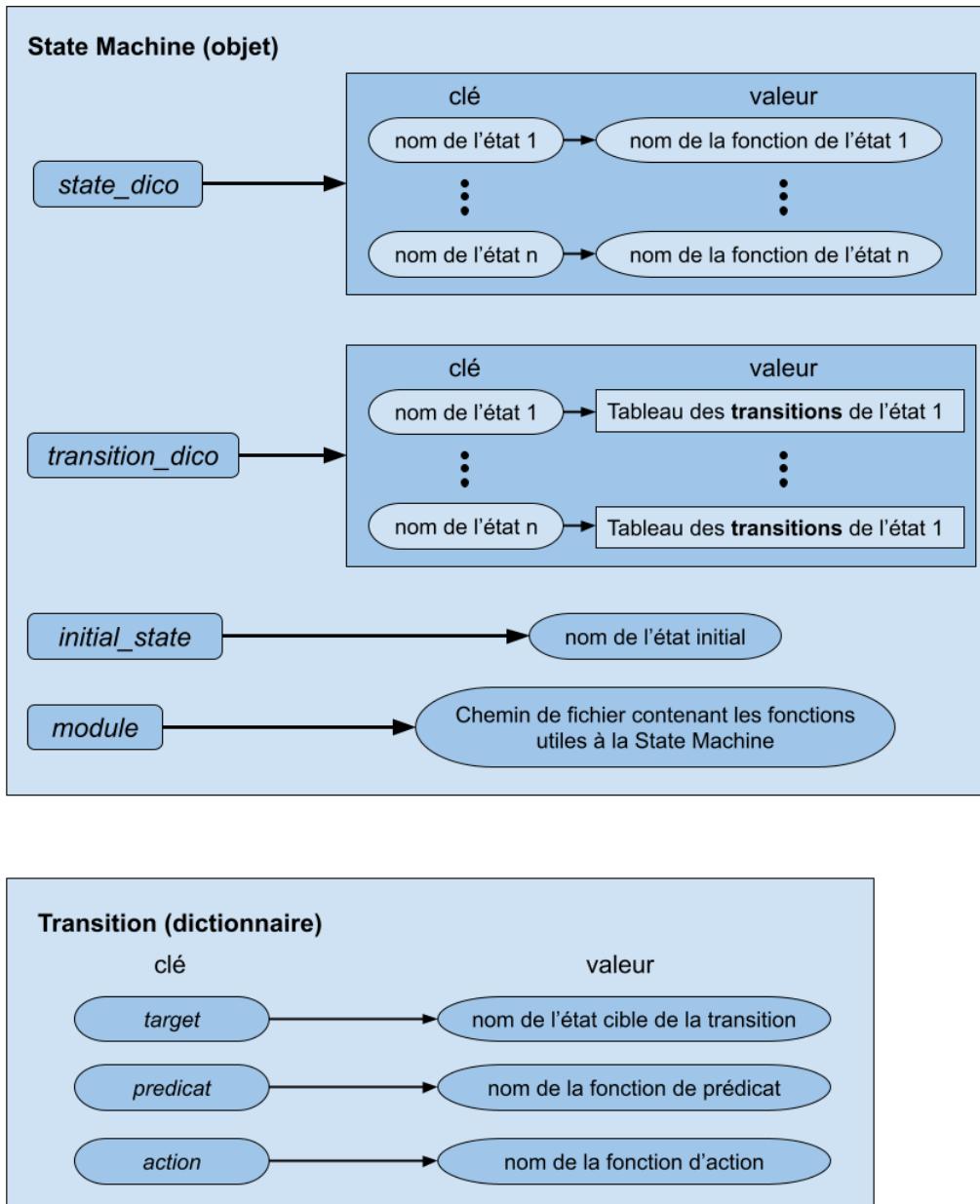


FIGURE 7 – Schématisation de la State_Machine

Nous voyons donc que seule la State_Machine est une classe à proprement parler tandis que les états et les transitions sont gérés par des dictionnaires et des tableaux. Dans ces différentes structures de données sont stockées des chaînes de caractères, il s'agit pour la plupart de noms de fonction qui sont ensuite appellées grâce à la fonction `getattr` de Python. Cependant, cette dernière a pour but de récupérer une fonction depuis une chaîne de caractères à l'intérieur d'un contexte. Or, la State_Machine ne possède pas encore les fonctions qu'elle va utiliser c'est pourquoi elle est munie d'un attribut `module` qui représente le nom du fichier dans lequel sont implémentées les fonctions utilisées par la State_Machine. Enfin, on trouve le champ `initial_state` qui est simplement le nom de l'état initial de la State_Machine.

Cela n'est pas directement mentionné dans le schéma mais la State_Machine embarque une `Session` afin de pouvoir interagir avec le Reachy. Cette notion de `Session` est discutée plus en détail dans la Partie 3.2.

3.3.3 Notion d'exécuteur

Une State_Machine comme présentée jusqu'à présent est un objet composé d'états et de transitions. Cependant, elle n'est pas faite pour être exécutée directement par elle-même, il s'agit davantage d'une structure de données qui englobe les états et les transitions. Pour cela nous avons ajouté une notion d'exécuteur, au départ un exécuteur prenait en argument une machine à états et l'exécutait mais suite à une discussion avec M. Morandat nous avons décidé d'une solution plus logique. La State_Machine est alors capable via une fonction,

de se créer un exécuteur. Celui-ci est représenté par la classe `Executor` et son but est de partir de l'état initial de la machine, de l'exécuter, de regarder les prédictats des différentes transitions sortantes de cet état afin d'en emprunter une jusqu'à l'état suivant et ainsi de suite. C'est donc l'objet qui va parcourir la `State_Machine` et plus précisément il s'agit du programme principal de Reachy. Afin d'avoir accès aux différents états et transitions, l'`Executor` possède un attribut enregistrant la machine à états qui l'a créé. Ainsi il l'utilise comme une structure de données afin de connaître les états, les fonctions associées, les transitions sortantes de chacun, etc...

L'un des principaux problèmes auxquels nous avons été confrontés afin de généraliser la machine à états a été le contexte d'exécution. En effet, une machine à états est un enchaînement d'états / transitions mais certaines valeurs doivent subsister aux états comme par exemple la `Session` qui est primordiale. C'est pourquoi nous avons ajouté un attribut à l'`Executor` nommé `context`. Il s'agit d'un dictionnaire, il est utilisé dans le fichier pointé par l'attribut `module` de la `State_Machine` dans lequel chaque fonction, dès lors qu'elle veut faire transiter de l'information au-delà de son propre environnement, place sa valeur dans le dictionnaire `context` à une certaine clé. Ainsi une fonction d'un autre état saura la retrouver en allant simplement accéder au dictionnaire `context`.

3.3.4 Fichiers JSON

Maintenant que nous avions généralisé la notion de machine à états, il a fallu trouver un moyen pour la création de ces objets `State_Machine`. Il aurait été possible de créer à la main les différents dictionnaires et tableaux dans un fichier. Cependant, cela paraissait peu pertinent puisque cela oblige les personnes souhaitant créer une `State_Machine` à connaître la structure en détail. Le choix s'est donc porté sur un stockage dans des fichiers `.json` et la création d'une fonction de chargement de ces fichiers pour les transformer en `State_Machine`. Nous avons choisi les `.json` tout d'abord car ils se portent bien à notre problème étant donné qu'il est facile de représenter des dictionnaires et des tableaux dans ce type de fichier. De plus, en `Python` il est d'autant plus simple de lire un fichier `.json` car une fonction existe pour le transformer directement en structure de données `Python`. La fonction de chargement, `loader`, se base donc sur cette fonction de `Python` afin de dégager du fichier `.json` le dictionnaire d'états, celui de transitions,etc... Ensuite la fonction construit une `State_Machine` à partir de cela et la renvoie. La Figure 8 illustre la structure du `.json` dont un exemple est présent en Annexe (cf 9.2).

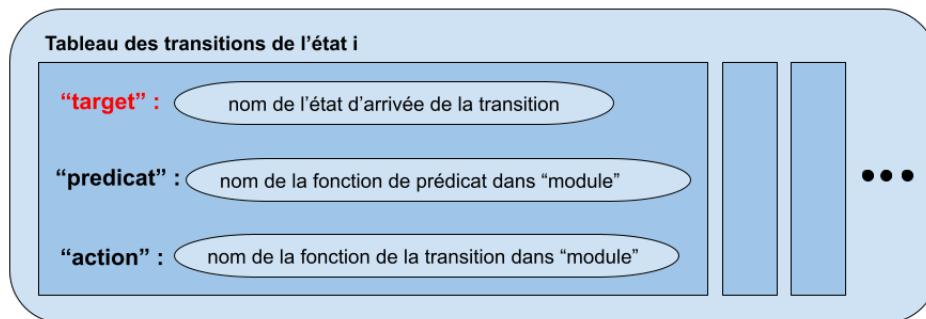
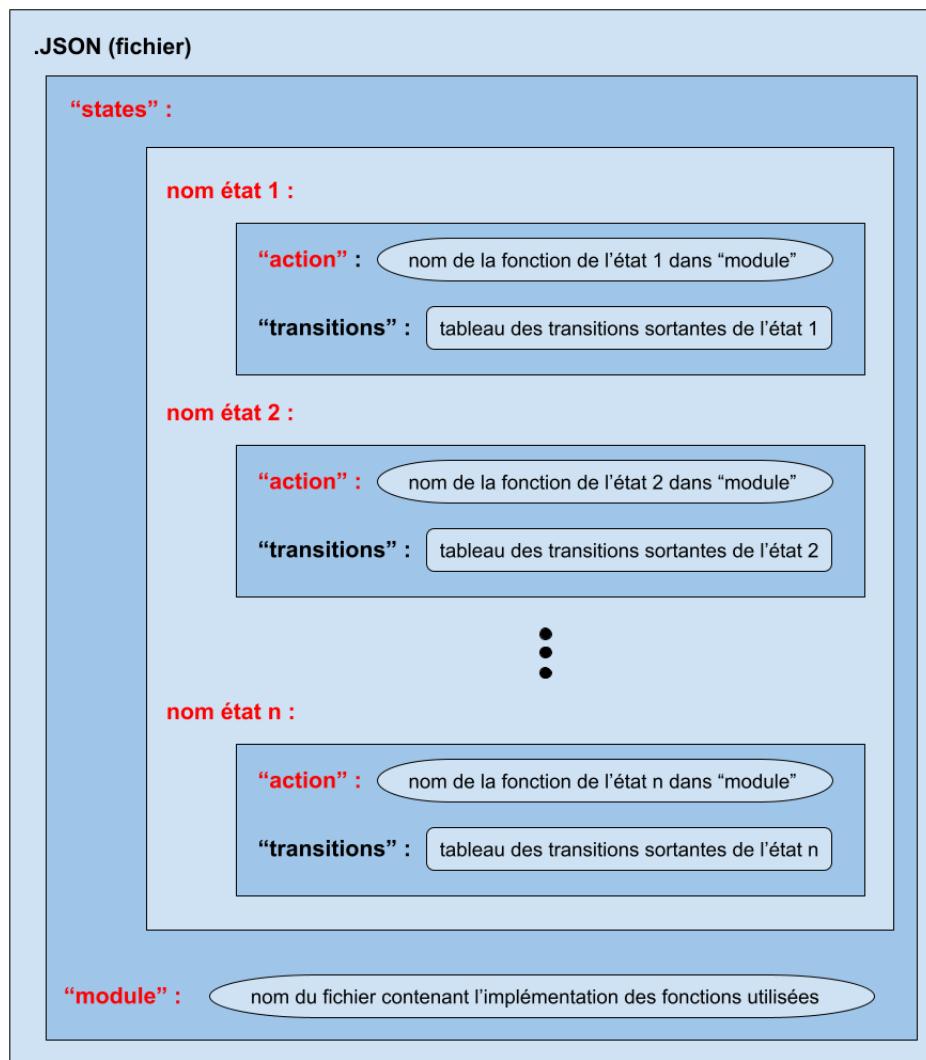


FIGURE 8 – Schématisation du format des fichiers .json

Nous remarquons donc sur la figure 8 que le format du fichier .json correspond plus ou moins à la structure de la machine à états. Néanmoins sur le schéma toutes les clés des dictionnaires ne sont pas toutes obligatoires et sont remplacées ensuite par l'Executor lors de l'exécution de la machine à états. Il s'agit des noms qui ne sont pas en rouge, ceux-ci sont obligatoires. Nous avons par exemple le prédicat d'une transition qui est facultatif et qui sera remplacé à l'exécution par une fonction qui renvoie toujours vrai. Nous pouvons donc assimiler cela à une transition par défaut. Il est important de noter que lors de la vérification des prédicats par l'Executor, celui-ci les vérifie dans l'ordre du tableau qui est le même que l'ordre dans le fichier .json. De ce fait, une transition dite par défaut par son absence de prédicat devra être placée à la fin du tableau pour ne pas rendre

d'autres transitions inaccessibles. De plus, d'autres clés sont optionnelles comme l'action d'une transition qui sera remplacée par une fonction ne faisant rien. Enfin, on trouve le tableau de transitions en clé optionnelle, un état dépourvu de transition sera considéré comme état final et arrêtera l'exécution de la machine à états.

3.3.5 Notion de Timeout pour un état

Après la généralisation de la machine à états, nous avons essayé de mettre en place une première machine de test pour le Reachy. La `State_Machine` de celui-ci comporte des `timeout` sur certains états qui obligent à changer d'état au bout d'un certain nombre de secondes. La première implémentation de cela s'est faite via des transitions dans la machine à états avec une fonction de prédicat qui vérifie le temps passé depuis le début de l'état (valeur sauvegardée dans le contexte). Si cette durée était supérieure à une certaine valeur alors la transition était prise. Cependant, cette méthode rend plus difficile l'écriture du fichier `.json` et de plus, chaque fois qu'un nouveau timeout devait être fait avec une durée en secondes différentes il fallait créer un prédicat différent ce qui n'est pas pratique lorsque plus de 3 ou 4 états ont besoin de ce système. Ce qui a été décidé pour régler ce problème est l'ajout d'une clé optionnelle "`timeout`" dans chaque état du fichier `.json`. La figure 9 schématisse cet ajout.

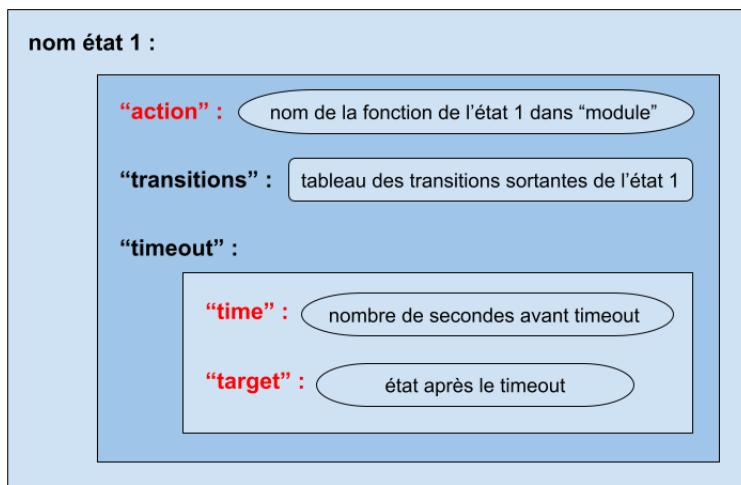


FIGURE 9 – Schématisation du format des timeouts dans les JSON

Nous pouvons voir sur la figure 9 que la clé "`timeout`" est optionnelle, si elle n'est pas renseignée l'exécuteur continuera à boucler sur l'état tant que celui-ci n'a pas de prédicat de transition vérifié. "`timeout`" est un dictionnaire composé de 2 clés, "`time`" qui est un entier représentant le nombre de secondes avant le timeout et "`target`" qui est l'état à atteindre en cas de timeout.

Cet ajout a fortement facilité l'implémentation de timeouts et de la machine à états en général qui pouvait devenir compliquée avec la gestion du temps depuis l'arrivée dans un état et tout ce que cela implique comme la réinitialisation du temps dans le contexte par des actions de transitions.

3.3.6 Lancer une machine à états sur le Reachy

Afin de lancer une machine à états sur le Reachy il suffit simplement de lancer un exécuteur de la `State_Machine` choisie puisque Reachy peut être assimilé à un exécuteur. Pour cela il faut créer une session (cf Partie 3.2), la donner à l'exécuteur et lancer ce dernier. Le Reachy va alors passer d'états en états en effectuant les actions associées.

Après la généralisation de la notion de machine à états, pour chaque `State_Machine` un fichier Python était créé afin de charger la machine, créer une session et lancer un exécuteur sur celle-ci. Cela commençait à prendre de la place et surtout à rendre illisible le fichier `src/state_machine`. C'est pourquoi nous avons implémenté un fichier qui utilise la ligne de commande pour charger le fichier `.json` voulu. Il en existe 2, les deux prennent en argument le chemin relatif vers la machine à états au format `.json` (cf Partie 3.3.4), la charge et l'exécute. Cependant, l'un le fait avec une session associée au Reachy tandis que l'autre le fait avec une fausse session afin de pouvoir tester les machines à états sans avoir besoin de se déplacer au FabLab.

En conclusion de cette partie sur la machine à états, sa généralisation nous a pris beaucoup plus de temps que la première implémentation cependant nous avons pu ensuite gagner énormément de temps. Nous avons même pu en quelques minutes préparer des State_Machine de tests et en 1 heure il a été possible d'écrire une machine à états pour la soirée partenaire avec l'utilisation exclusive de codes ARUCO.

4 Les différentes fonctionnalités

Pour permettre au robot d'interagir avec les utilisateurs, il fallait tout d'abord rendre possible la reconnaissance vocale.

4.1 Reconnaissance vocale

Toute la partie reconnaissance vocale a été implementée grâce aux bibliothèques PyAudio pour la récupération du son ainsi que de speech_recognition pour la partie traduction de vocal à texte. Cela correspond au cahier des charges.

4.1.1 Gestion des mots-clé

La partie reconnaissance vocale devait dans un premier temps être capable de reconnaître des mots-clés et de réagir en fonction. Pour la partie ordre, la reconnaissance d'un mot-clé doit déclencher un changement d'état et cela est vérifié par les prédictifs présentés dans la partie 3.3.2. Pour la partie conversation, la reconnaissance des mots-clés se traduit par un ensemble de mots d'entrée qui donnent lieu à des mots-clés de sortie. Cela a été mis en place grâce à des fichiers .txt comportant, comme spécifié dans le cahier des charges, un ensemble de mots de sortie et un ensemble de mots d'entrée. Dans un fichier Python nous avons implementé des fonctions qui ouvrent ces fichiers et en créer des tableaux avec l'ensemble des mots-clés. Ensuite, dans la machine à états, il y a une comparaison entre le texte obtenu par la fonction de reconnaissance vocale et les tableaux afin de savoir quels mots sont présents et en déduire une sortie. Un principe similaire a été appliqué pour les mots-clé donnant lieu à un changement d'état. La Figure 10 schématisse les différents éléments qui entrent en jeu dans le système de reconnaissance vocale.

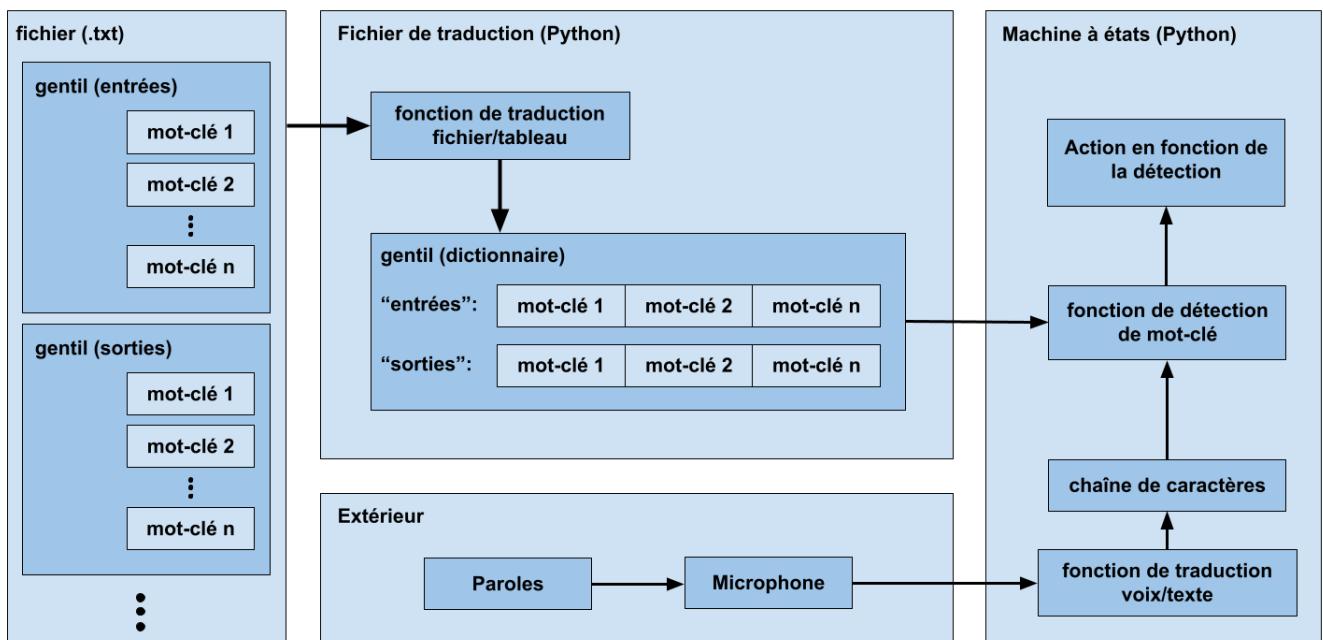


FIGURE 10 – Schématisation des étapes de la reconnaissances de mots-clé pour le conversation

Nous remarquons sur la figure 10 que les entrées et sorties des conversations sont gérées par des dictionnaires ce qui rend l'utilisation des couples entrée/sortie plus simple.

Toutes ces interactions entre fichiers, fonctions de traduction et machine à états viennent d'une réflexion afin de rendre l'utilisation de mots-clés plus accessibles. En effet, de cette manière pour ajouter un mot-clé à un couple entrée/sortie ou bien un mot-clé pour passer de l'état Attente d'ordre à Photo par exemple, il suffit simplement à l'utilisateur d'ajouter un mot dans un fichier texte sans toucher au code. Cela paraissait

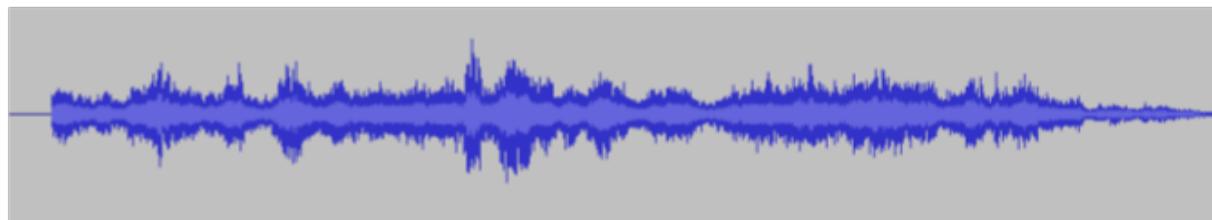
beaucoup plus instinctif. Cependant, si l'utilisateur souhaite créer un nouvel ensemble de mots-clés il est obligé d'effectuer quelques modifications dans le code. Au moins pour le système d'entrée/sortie, il aurait été intéressant d'automatiser tout ce processus pour permettre à l'utilisateur d'ajouter des couples d'entrée/sortie sans toucher à une seule ligne d'un fichier Python. Cela aurait également permis de factoriser du code mais on aurait perdu le fait d'ajouter des mouvements de réaction aux mots-clés pour le Reachy. La solution actuelle semble donc convenir dans la mesure où les modifications sont minimales et que l'on gagne en logique dans la machine à états.

4.1.2 Gestion prise du son

Afin d'enregistrer la voix de l'interlocuteur avec le robot nous avons utilisé le microphone interne à celui-ci. Cependant nous n'avons pas réussi à trouver des réglages convenables, il fallait donc la plupart du temps parler très fort et lorsqu'il y avait du son parasite il était impossible de lui faire détecter des mots-clé. Même avec l'utilisation de fonctions de réduction du bruit, il était impossible d'obtenir un résultat convenable. Nous avons essayé de changer de microphone en lui branchant le même modèle que son micro interne mais cette fois-ci à l'extérieur mais le problème était le même. Le problème venait donc du modèle du microphone et non pas du microphone interne en particulier. Nous avons ensuite essayé avec un casque-micro et cette fois-ci aucun problème de compréhension même dans un environnement moyennement bruyant. La solution que nous avons trouvée est donc d'utiliser le micro du casque-micro pour l'entrée audio et les hauts-parleurs du Reachy pour la sortie audio.

4.1.3 Réduction du bruit

Pour ce qui est de la réduction du bruit, il était mentionné dans le cahier des charges que nous devions utiliser un système afin d'obtenir une entrée avec des bruits parasites réduits pour le Reachy. Nous avons réalisé cela grâce à une bibliothèque Python et après les tests avec le casque-micro cela fonctionne correctement. Afin de vérifier ce bon fonctionnement nous sommes passés par des sous-fichiers, c'est-à-dire que nous enregistrons la voix brute dans un premier fichier puis nous effectuons un traitement sur celui-ci et enregistrons le résultat dans un second fichier. Cela nous a permis de les comparer et nous avons observé une amélioration au niveau des bruits parasites. La figure 11 illustre cette variation, pour obtenir cette courbe nous nous sommes enregistrés disant une commande au Reachy dans des conditions de bruit relativement élevées.



(a) Sans réduction



(b) Avec réduction

FIGURE 11 – Visualisation de la différence avec ou sans la fonction de réduction du bruit

4.1.4 Conversation avancée

Le robot doit être capable de mener une conversation soit du type simple ou du type avancée. Dans le cahier des charges, nous avons mentionné que la conversation avancée allait être implémentée de la même façon que

la conversation simple, c'est-à-dire, en utilisant des mots-clés et en ajoutant une notion de mémoire de l'état à la conversation. Nous nous sommes rendus compte, que la conversation allait rester toujours très limitée. Alors, nous avons décidé d'utiliser **OpenAI**.

OpenAI est une API qui permet d'accéder à GPT-3 une intelligence artificielle qui effectue diverses fonctionnalités du traitement du langage naturel (ou en anglais Natural Language Programming NLP). Il y a quatre modèles (ou engines) dans GPT-3 : "text-davinci-002", "text-curie-001", "text-babbage-001" et "text-ada-001". Chacun de ces modèles a des caractéristiques et des niveaux de performances destinées à des tâches spécifiques. Nous avons choisi "**davinci**" parce qu'il est le modèle le plus puissant et parce que l'API est en version **beta**. Ainsi, pour utiliser cela, nous avons dû fixer quelques paramètres :

- température : La température est un paramètre qui permet de contrôler la diversité des réponses et qui donne un effet de l'aléatoire. Il est compris entre 0 et 1. Plus on s'approche de zéro, plus les réponses sont déterministes et répétitives. Choix : 0.9
- max_tokens : est le paramètre qui fixe la taille en nombre de mots de la réponse en sortie. Nous l'avons fixé à 100 pour que le robot ne donne pas des réponses trop longues et ennuyeuses pour l'utilisateur.
- best_of : L'algorithme génère, pour chaque entrée, plusieurs réponses avec une certaine probabilité. Ce paramètre permet de fixer le nombre des meilleures sorties (nous avons choisi de le mettre à 1) pour avoir seulement la réponse avec la plus grande probabilité.
- top_p : La probabilité utilisée pour filtrer les réponses. Si elle est à 50% , alors la moitié de toutes les options pondérées probables sont considérées pour filtrer la réponse. (choix : 1)
- frequency_penalty : permet de pénaliser les nouveaux "tokens" ou mots en se basant sur leur fréquence de présence dans la conversation. Cela permet d'éviter la répétition. (choix : 0)
- presence_penalty : permet de pénaliser les nouveaux tokens s'ils ont apparu dans la conversation. Cela permet d'encourager le modèle à parler de nouveaux sujets. (choix : 0.6)

Après l'intégration de cette fonctionnalité, nous avons rencontré un problème de langues. En effet, GPT-3 est censé parler en anglais comme langue par défaut. Et puisque le choix de la voix était en français, c'était très difficile de comprendre les réponses du robot. Et d'une manière générale, le fait d'avoir des entrées dans une langue et la sortie dans une autre langue diminue nettement la performance. La solution initiale que nous avions trouvée est l'ajout de certaines phrases d'initialisation écrites en français pour le forcer en quelque sorte à parler en français. Mais après l'ajout de l'état incompréhension "qui mène vers la conversation avancée", nous avons décidé de lui passer en paramètre un **context** qui lui permet d'enregistrer l'état de la conversation avec l'utilisateur avant le début de la conversation avec l'IA et donc d'initialiser une conversation en français dans ce **context**.

4.2 Synthèse vocale

L'une des fonctionnalités de Reachy requise dans le cahier des charges est de parler et d'interagir avec l'utilisateur. Tout le code de la partie synthèse vocale est présent dans le répertoire **speech** de **src**. Il est séparé en deux fichiers principaux. Le fichier **speech_synthesis.py** qui implémente une première utilisation de la bibliothèque Python **pyttsx3**. Un deuxième fichier **speech_synthesis_gtt.py** qui lui implémente la solution maintenue qui repose sur l'utilisation de l'API gTTS.

4.2.1 Utilisation de **pyttsx3**

Le module **pyttsx3** est une bibliothèque de conversion de texte en parole en Python12. Sa particularité principale est son fonctionnement hors ligne. Il supporte deux voix : la première est féminine et la seconde est masculine. Il prend en charge 3 moteurs TTS : **sapi5**, **nsss** et **espeak**. Nous avons utilisé le moteur espeak, comme indiqué dans le cahier des charges, vu que les deux autres moteurs TTS ne peuvent pas être lancés et ne sont pas pris en charge sous Linux.

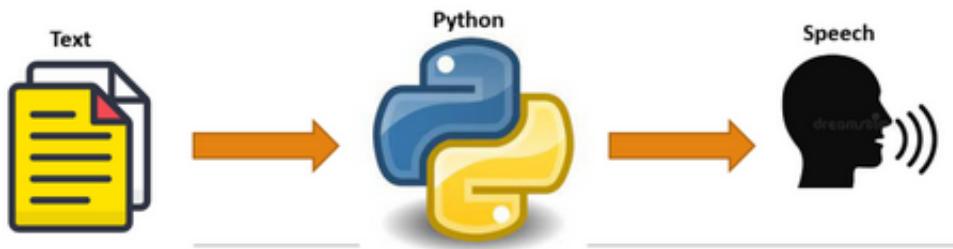


FIGURE 12 – Synthèse vocale

Tout le code qui utilise le module `pyttsx3` a été implémenté dans le fichier `speech_synthesis.py` du répertoire `speech`. On y contrôle directement le volume, la vitesse et la voix.

Cependant, malgré le bon fonctionnement du code, la voix générée reste très mécanique. Et ceci s'explique par le fait que nous utilisons le synthétiseur vocal `espeak`. La seule option pour améliorer et changer la voix générée par le module `pyttsx3` était de sélectionner l'un des deux autres moteurs TTS. Cependant, cela était impossible à réaliser puisque les deux autres synthétiseurs ne sont pas pris en charge sous Linux.

4.2.2 Utilisation de gTTS

Une des solutions que nous avons envisagées pour avoir une voix robotique mais pas trop mécanique est d'utiliser un autre module de synthèse vocale. C'est pourquoi nous avons choisi d'utiliser le module `gTTS`, Google Test-to-Speech, qui utilise l'API de synthèse vocale de Google Traduction. Cet outil est en ligne et nécessite donc un accès à Internet. Ce moteur propose une voix féminine et non modifiable. Néanmoins, nous avons trouvé que la voix proposée n'est ni humaine, ni très mécanique. Elle correspond parfaitement à la voix d'un petit robot.



FIGURE 13 – gTTS pour la conversion d'un texte en audio

L'API propose différentes langues pour la lecture du texte en entrée. Nous avons décidé de fixer ce paramètre et de choisir le Français comme langue de communication entre l'utilisateur et le robot.

Déroulement du script : À l'exécution du script, la fonction `text_to_speech` prend en entrée un texte, génère la voix associée dans un fichier `.mp3` temporaire dans le répertoire `tmp/`, lit le fichier et le supprime directement après pour optimiser l'espace mémoire. À la lecture du fichier, le son sort directement de la sortie audio du robot.

Transcription d'une émotion ou d'un état : Il a été spécifié dans le cahier des charges d'émettre des sons et des bruits suivant les émotions et les états du robot. Pour ce faire, différents sons ont été mis dans le dossier `voices/` sous forme de fichiers `.mp3`, et chaque son est émis suivant l'état du robot, lorsqu'il prend une photo par exemple, il compte jusqu'à trois et émet un son de flash, comme si la photo était prise par un appareil photo.

Structuration du code : Pour certains états et transitions de notre machine à états, le robot doit émettre du son, soit pour communiquer avec l'utilisateur ou pour exprimer ses émotions via de petits bruits. Pour cela, nous avons consacré une fonction pour chaque état ou transition, ayant besoin de la synthèse vocale, pour sélectionner le son associé.

Amélioration possibles :

- Nous pouvons imaginer un système qui vérifie si le robot est connecté à internet ou pas. La réponse renvoyée décidera si le robot utilisera la version connectée `gTTS` ou la non connectée `pyttsx3`.
- Une deuxième amélioration possible serait de laisser la possibilité à l'utilisateur de configurer la langue avec laquelle il aimerait interagir avec Reachy.

4.3 Traitement d'image

L'une des grandes fonctionnalités de Reachy requise par le client est la capacité à prendre des photos "bien" cadrées, ainsi que d'apporter certaines retouches aux photos prises mais aussi de suivre une personne du regard pour rendre l'interaction plus naturelle. Pour ce faire et respecter les contraintes fonctionnelles, nous devions implémenter la capacité d'identifier des visages (au sens de la détection) et de cadrer les personnes souhaitant être prises en photo, avant de considérer l'ajout de filtres sur les photos en questions.

Ces services sont implémentés dans le fichier `face_detection.py`. L'architecture de ce code source a été pensée de façon modulaire en offrant une interface pour le recours aux dits services, mais est également pensée de façon fonctionnelle et en couche¹. Ces propriétés permettent une maintenance et une amélioration facile du code en plus de permettre la composition de plusieurs services afin d'offrir une explosion combinatoire de transformations d'images.

4.3.1 Prise de photos

Détection de visages

La première problématique a été l'intégration de la capacité à détecter des visages. La détection faciale étant une branche importante de l'intelligence artificielle, la méthode la plus efficace pour recourir à un algorithme performant consista à exploiter une bibliothèque implémentant des services de vision par ordinateur.

OpenCV est une bibliothèque proposant de nombreuses fonctionnalités de vision par ordinateur et spécialisée dans le traitement d'images en temps réel. C'est une bibliothèque très médiatisée, facile d'utilisation et que certains membres de l'équipe ont déjà manipulée. Celle-ci nous a permis d'économiser du temps sur les détails techniques de l'implémentation des méthodes permettant de fournir les services requis. Qui-plus-est, cette bibliothèque nous a été conseillée par le client pour ces mêmes raisons pratiques.

Le port du masque

La première implémentation consista en une interface exploitant l'une des caméras² de Reachy pour identifier des visages dans son champ de vision et proposer des angles de rotation en repère sphérique à l'interface de mouvement afin d'obtenir une première approche du suivi de visage et d'un cadrage. Le problème majeur inattendu lors de la programmation et tests à distance de cette implémentation fut la présence de masques contre la COVID-19. Ceux-ci cachaient une partie conséquente des visages ce qui entravait les capacités de détection de visages de certaines méthodes cherchant la présence d'un nez et d'une bouche pour fonctionner. Cette découverte lors d'un test au Fablab a restreint la gamme de méthodes de reconnaissance faciale utilisables.

En conséquence, la méthode la plus connue et résistante au port du masque est la **classification en cascade de Haar**. Grâce à des fenêtres de détection³ permettant d'analyser localement la répartition des couleurs sur une image, la méthode de Viola et Jones exploitée dans cette classification en cascade permet de reconnaître de nombreux objets et formes. Cette propriété permet, en possession d'un classificateur entraîné, de reconnaître uniquement la partie frontale d'un visage. Ceci nous a permis de recourir à une méthode fonctionnant avec ou sans port du masque grâce à la détection des yeux et du front en tant que détection faciale. Ce problème n'a pas été handicapant étant donné la modularité de la bibliothèque **OpenCV** permettant aisément de changer de méthode de reconnaissance faciale au sein du code source.

Tangage de la tête

Un problème paraissant en tant que cette technique fut la difficulté avec laquelle la tête de Reachy se mouvait sans induire un penchement de la tête. Ce problème de tangage fut corrigé par les membres de l'équipe gérant les mouvements. Cependant, sans une telle correction, l'analyse d'image en vu de fournir des angles de déplacement de la tête aurait été mise en défaut.

Le problème venait de l'interprétation qu'avait Reachy de son environnement visuel. Avec une tête penchée comme montré en figure 14, Reachy conçoit les axes vertical et horizontal selon le référentiel de sa caméra. Or, avec une tête penchée, ce référentiel n'est plus confondu avec celui de la salle formant son environnement. Cette

1. Les couches sont : transformations mathématiques en espace plan, traitement de tableaux, sous fonctions offrant des traitements paramétrables à l'interface, fonctions d'interaction avec les sessions, fonctions faisant office d'interface

2. Nous aborderons dans la suite la raison d'un tel choix

3. Ce sont des matrices parcourant une image afin d'associer des valeurs à des portions de l'image pour identifier des schémas particuliers

déformation visuelle nous a empêché d'effectuer des tests corrects de suivi de visages et de cadrage de photos. Cela s'explique par le fait que les angles de déplacement offerts par l'interface de vision par ordinateur étaient penchés par rapport à ceux que l'interface de mouvement utilise (fidèles au repère de l'environnement).

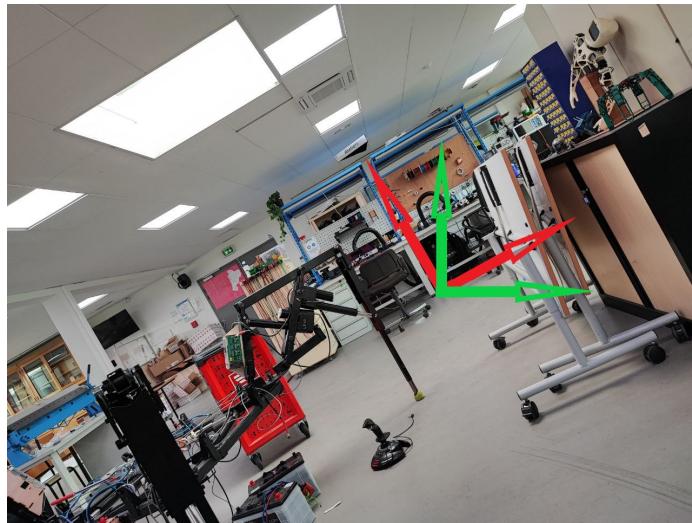


FIGURE 14 – Exemple de décalage entre les référentiels rouge (de la salle) et vert (de la caméra)

Avant la découverte d'une correction de ce problème au niveau de l'interface de mouvement, nous avons conçu diverses stratégies inefficaces ayant pour objectif d'offrir une correction au niveau de l'analyse d'image même.

La première fut l'exploitation de capteurs propres à Reachy permettant d'avoir une connaissance du tangage de la tête pour apporter une rectification des angles de mouvement au travers d'une transformation de ceux-ci. Cependant, la manipulation des moteurs étant très bas niveau, cette solution n'a pas été retenue pour sa complexité de développement.

La seconde alternative aurait été l'analyse d'une ligne d'horizon dans l'image même afin de déterminer le tangage et d'apporter les corrections d'angles. Mais Reachy étant posé sur une table, celui-ci voit son champ de vision souvent en contre-plongée, empêchant ainsi de voir le sol qui aurait servi à définir l'horizon. La rectification dans l'interface du mouvement a été apportée assez rapidement pour que nous n'explorions pas d'autres pistes de correction.

Une ou deux caméra(s) ?

Une nouvelle entrave au suivi de visages était dans le recours à une caméra en particulier. En effet, Reachy possède deux caméras d'avantages espacées que des yeux humains. Nous avions une appréhension quant à la façon avec laquelle Reachy regarderait un individu.

Ainsi, nous avions pensé à utiliser une méthode de transformation linéaire afin de coupler la vision des deux caméras en une seule image pour prôner la cohérence avec laquelle Reachy fixerait quelqu'un. Ceci n'étant pas problématique pour le cadrage de photo, et étant donné que la caméra regardant de face un individu est celle prenant les photos, le client et nous-même n'avons pas jugé pertinent de consacrer du temps à une telle correction. De plus, lors de nos premiers tests de suivi de visages, la façon dont le robot ciblait une personne du regard était tout à fait naturelle.

Cadrage

Concernant le cadrage des photos selon les visages à capturer, celui-ci se décompose en deux catégories. La première est le cadrage d'une photo simple, c'est-à-dire une photo ne comprenant qu'une seule personne, tandis que le second type concerne la prise de photos de groupe.

Après connaissance du type de photo à effectuer, le robot ajuste la position de sa tête en conséquence afin de cadrer la photo. Dans le cas particulier d'une personne seule, nous avons convenu avec le client qu'il est naturel de centrer la photo sur le centre du visage de la personne concernée comme montré en figure 15. Afin de s'assurer que d'autres visages n'interfèrent pas avec celui de la personne à cadrer, un traitement de l'image identifie le visage le plus proche de Reachy et considère ce visage comme étant celui de l'utilisateur actuel. Cette politique a été approuvée par le client.

Cependant, la notion de proximité est complexe dans le domaine de la vision par ordinateur. Des méthodes recourant à de l'intelligence artificielle permettent d'avoir une estimation de la distance d'un visage à la caméra mais ces méthodes sont très versatiles selon la luminosité et la présence d'obstacles à proximité du visage induisant une distorsion des distances parasitant la distance prédictive. Nous avons donc choisi, avec l'accord du client, d'approximer la notion de distance à la longueur d'un visage. Ceci permet de connaître une distance de façon déterministe et faiblement impactée par la différence de taille de visages.



FIGURE 15 – Placement du barycentre (point rouge) pour le cadrage d'une personne

Le cas d'une photo de groupe demanda d'avantages de réflexion sur la façon d'éliminer les visages parasites mais aussi de cadrer d'une façon satisfaisante pour le client.

La politique d'élimination des visages parasites n'a pas reçu de contraintes venant du client. Une façon d'aborder ce souci s'est portée sur le recours à la proximité des visages⁴. Après identification de tous les visages dans le champ de vision de Reachy, une analyse de la proximité moyenne est effectuée. Avec une paramétrisation de l'écart à cette moyenne permise pour les visages présents, il nous a été possible d'éliminer les visages trop loin ou trop proches de cette moyenne, étant jugés comme n'appartenant pas au groupe⁵ qui représente la majorité des visages à prendre en compte, donc possédant la plus grande pondération dans le calcul de la moyenne. Avec plusieurs essais (facilités grâce à la paramétrisation), il nous a été possible de calibrer cette élimination de façon subjectivement satisfaisante.

Enfin, une fois les visages indésirables enlevés de l'ensemble à considérer, le cadrage peut être effectué sur l'échantillon de visages restants. La façon dont le cadrage de groupe est réalisé a demandé quelques échanges avec le client afin de choisir une façon parmi deux proposées.

La première façon était de considérer la même distance séparant le visage le plus à gauche de la limite gauche du champ de vision de Reachy de la distance entre le visage le plus à droite et la limite droite du champ de vision de Reachy. Cette méthode n'était pas concluante dans la mesure où une personne éloignée du groupe sur le côté du champ de vision de Reachy (mais à la même distance de Reachy que les membres du groupe à prendre en photo) aurait joué un rôle indésirable et trop important dans le cadrage.

La seconde approche a été approuvée par le client et ainsi retenue. Celle-ci consiste à calculer le barycentre des points incarnant les centres des visages à cadrer. De ce fait, le cas précédent d'une personne sur le côté sera en partie corrigé avec une préférence d'orientation de la tête vers la majorité. Le résultat d'un tel calcul est visible

4. toujours approximée par la hauteur des visages

5. Considéré comme un ensemble de visages ayant une distance assez homogène à Reachy

en figure 16.



FIGURE 16 – Placement du barycentre (point rouge) pour le cadrage d'un groupe

La mise au point

Un problème handicapant la faculté visuelle de Reachy fut la gestion de la mise au point. Celle-ci peut être demandée via l'interface de Reachy mais la mise au point automatique était compliquée à maîtriser. Au départ, nous utilisions la mise au point avant chaque prise de photo afin que ladite photo soit nette. Or, l'échec de la mise au point rendait les photos floues. De plus, après échange avec le client, nous nous sommes rendu compte qu'une seule mise au point au lancement de Reachy était suffisante quelque soit la distance des personnes prises en photo par rapport à la caméra. Ainsi, afin de corriger le placement de la focale de la caméra dès que nécessaire, un programme est exécutable en parallèle de la machine à états. Ce programme affiche les images de la caméra en temps réel et lance la mise au point tant qu'une touche du clavier n'est pas pressée. Ceci permet de stopper la mise au point manuellement dès que les images envoyées par la caméra sont nettes.

Toutes les fonctionnalités concernant les manipulations visuelles de Reachy avant les 100 ans de l'Enseirb-Matmeca ont pu être implémentées fidèlement aux demandes du client répertoriées dans le cahier des charges.

4.3.2 Filtres

Deux filtres ont été proposés et retenus par le client. Le premier consiste à retirer les couleurs d'une photo afin de la proposer en noir et blanc. Le second filtre, requérant une étude plus poussée quant aux attentes du client, était d'effectuer un échange de visages.

La coloration en noir et blanc a été très rapide étant un service déjà fourni par la bibliothèque **OpenCV**.

L'échange de visages est une retouche d'image très médiatisée et appréciée du client. Différentes approches existent afin d'obtenir des résultats plus ou moins poussés. Afin de satisfaire le client, ayant affirmé sa satisfaction sur une échange basique de deux visages par 'copié-collé' directe des zones, une méthode avancée d'échange de

visages a été implémentée, la **triangularisation de Delaunay**.

Cette triangularisation s'appuie sur la présence de points caractéristiques sur le visage tels que le bout du nez, les extrémités des paupières ou bien de la bouche comme illustré en figure 17. Ces points permettent de définir les côtés des triangles qui définiront le partitionnement du visage en surfaces élémentaires comme montré en figure 18.

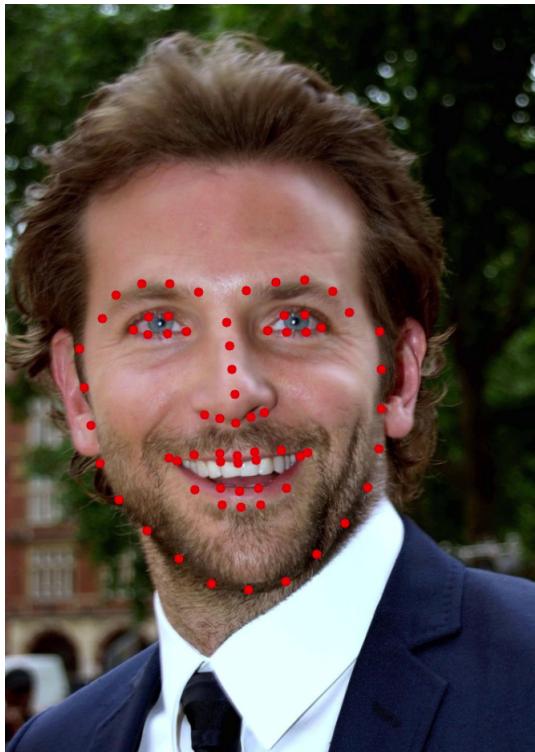


FIGURE 17 – Placement des points caractéristiques du visage

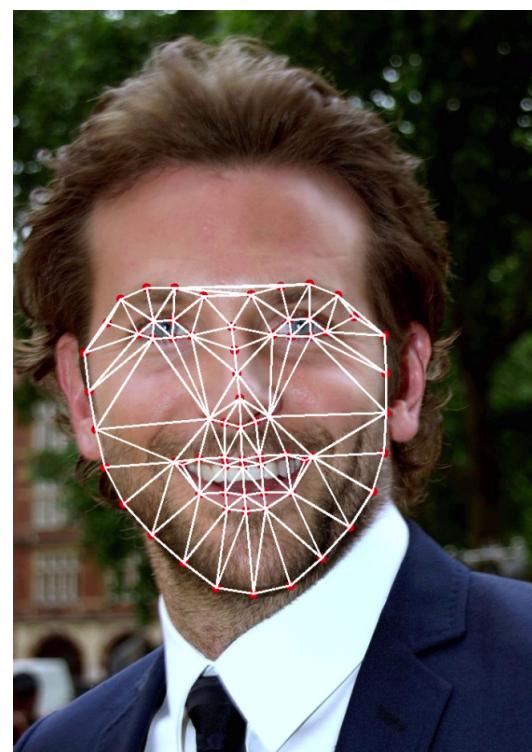


FIGURE 18 – Placement des triangles selon les points caractéristiques

Selon le cahier des charges, l'échange de visages aurait dû recourir à nouveau à **OpenCV**. Mais le recours à un algorithme identifiant les points caractéristiques d'un visage n'a pu se faire avec **OpenCV**. C'est donc la bibliothèque **dlib**, massivement utilisée pour des transformations faciales, qui a été utilisée.

C'est grâce à la **classification linéaire SVM** (Support Vector Machine) de la bibliothèque **dlib** que le placement des points sur un visage est déterminé.

Le partitionnement du visage en triangles permet d'adapter le visage à copier selon l'expression du visage à remplacer comme l'illustre la déformation linéaire d'un triangle en figure 19. Cela permet d'obtenir une meilleure cohérence quand à l'échange via une conservation des traits faciaux du visage de destination.

De ce fait, on peut observer le résultat d'une application de ce filtre en figures 20 et 21. Cette façon d'échanger les visages possède cependant un défaut concernant certaines expressions antagonistes telles qu'un visage avec la bouche ouverte et le second avec la bouche fermée. Le programme essaye d'adapter la texture de la bouche fermée afin de simuler la personne souriante, ce qui n'est pas esthétique (il en est de même avec les yeux ou le port d'accessoires). Qui-plus-est, la difficulté des transformations n'a permis que l'échange de deux visages. Ce sont les deux visages les plus proches qui seront pris en considération pour cet échange.

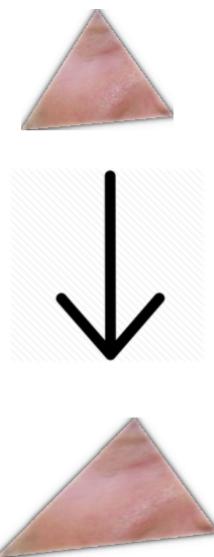


FIGURE 19 – Déformation linéaire d'un triangle



FIGURE 20 – Photo avant échange de visages



FIGURE 21 – Photo après échange de visages

Une dernière fonctionnalité évoquée avec le client comme étant optionnelle consistait à donner un indicateur de similarité entre deux visages. Cependant le manque de temps a empêché le développement de cet objectif avancé cité dans le cahier des charges.

Ces deux filtres sont de plus utilisables aisément à tout endroit où une image est manipulée étant donné que les fonctions offrant ces services prennent une image en entrée et ressortent la même image modifiée. Cette modularité permet d'utiliser les filtres aisément dans le code source des machines à état, y compris pour appliquer une modification directement sur la capture vidéo (une proposition d'application avancée de filtres).

4.3.3 Gestion des photos : Enregistrement, affichage et suppression

Pendant des événements de démonstration des capacités de Reachy, il nous sera utile de présenter aisément les photos récemment prises en les affichant de façon automatique sur un écran. Il faut également s'assurer de la sauvegarde des photos, étant une fonctionnalité requise par le client. De plus, les photos sont un point central parmi les données persistantes en mémoire que Reachy devra gérer. Étant donnée que Reachy est doté d'une mémoire finie, la gestion de l'accumulation des photos est un prérequis indispensable ayant besoin de définir une politique de suppression afin de déléguer cette gestion à un programme qui nous permettra, lors d'événements, de nous focaliser sur les démonstrations du robot.

A chaque fois qu'une photo est prise, celle-ci est automatiquement enregistrée dans le dossier `tmp/img` servant de lieu de stockage des photos. Afin d'avoir un suivi temporel des photos, le nom des photos donné à l'enregistrement est sous la forme : `année_mois_jour_heure_minute_seconde.png`. Cependant, cette information est à titre indicatif pour avoir un suivi visuel des moments où les photos sont prises. C'est grâce à la bibliothèque `os` que nous aurons connaissance des dates de création des photos afin

d'apporter les traitements désirés.

Afin d'offrir un affichage des photos pendant les démonstrations de Reachy, un programme dédié est lancé en parallèle, en tant que processus indépendant de celui faisant fonctionner la machine à états. Celui-ci exploite la bibliothèque **os** afin de repérer quelle photo présente dans **tmp/img** est la plus récente et affiche en temps réel la photo trouvée sur un écran relié à Reachy par un câble HDMI grâce à une fonctionnalité de la bibliothèque **OpenCV**.

Cela permet aux utilisateurs de vérifier si leur photo est convenable afin de demander en directe à Reachy d'en prendre une nouvelle dans le cas non échéant. Cette fonctionnalité est exécutée en parallèle du programme de la machine à états afin d'avoir une meilleure maîtrise en temps réel de ce programme mais aussi par soucis de complexité du code quand à son insertion dans une machine à états.

De façon similaire au fonctionnement du programme précédent, la politique de suppression des photos est exécutée en temps réel par un processus fonctionnant en parallèle. La mémoire de Reachy étant limitée, nous nous sommes assuré de déléguer la tâche de suppression en temps réel d'un trop plein de photos à un programme. Celui-ci exploite également la bibliothèque **os** pour classer les photos présentes dans **tmp/img** par date de création et supprime les photos les plus anciennes.

Cette politique de suppression a été approuvée par le client. De plus, la fonction offrant ce service est paramétrable de telle façon que nous devons indiquer le nombre de photos maximum à partir duquel la suppression sera effectuée mais aussi le nombre de photos qui seront supprimées à chaque appel de cette fonction.

Ceci permet par exemple d'indiquer une limite de 500 photos et de demander la suppression de 50 photos en cas de dépassement de ce seuil. Ainsi, à 501 photos, la fonction en supprimera 50 et il en restera 451 de façon à ce que ce traitement ne soit appelé qu'après la création de 50 nouvelles photos (ce qui permet de temporiser l'appel à cette fonction).

Cette tâche pouvant être lourde selon le nombre de photos à traiter, nous avons prévu une attente passive de 10 secondes réveillant le processus via un signal grâce à l'utilisation de la fonction **sleep()** (temps jugé pertinent au vu de la faible fréquence de prise de photos de Reachy).

Il était spécifié dans le cahier des charges que lors d'un dépassement de la mémoire, un signal serait envoyé au administrateur afin de supprimer les photos par lancement manuel d'un programme et selon la même politique que le programme précédemment présenté. Cette autonomie, appréciée par le client, nous a permis de gagner en facilité de gestion de Reachy lors d'événements.

4.4 Détection des tags Arucos

Les codes Aruco sont un pilier du fonctionnement de Reachy en cas de dysfonctionnement de la reconnaissance vocale et se sont révélés indispensables lors de la soirée partenaire. Ce sont des carrés similaires à des QR codes, avec un nombre de pixels généralement plus restreint et ayant pour objectif d'encoder un nombre entier naturel comme illustré en figure 22. Ces codes sont souvent utilisés pour identifier des objets facilement à partir d'un traitement d'image rapide.

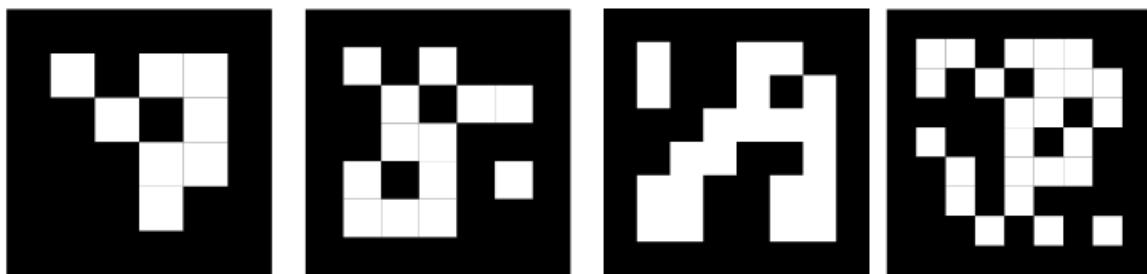


FIGURE 22 – Gamme de codes Aruco

Pour notre projet nous avons décidé d'utiliser des Arucos de taille 4. Le fonctionnement de la détection et reconnaissance des codes Aruco est très similaire à celui des visages. Après identification de la surface plane définissant le code, la dimension de la matrice associée est déterminée et le calcul de la valeur représentée est effectué. Pour effectuer cela nous utilisons l'api **aruco** de **OpenCV**, celle-ci utilise un dictionnaire **MARKER_DIC** pour connaître l'ensemble des Arucos possible ainsi que la fonction permettant de détecter les Arucos **PARAM_MMARKERS**.

L'association entre des nombres et des transitions nous a permis de faire passer Reachy d'un état à un autre grâce à ses facultés visuelles. Cependant, le nombre de codes utilisés a impliqué une restriction des interactions orales possibles quant aux conversations.

4.5 Mouvements

En plus de la fonctionnalité d'assistance vocale et de prise de photo, le robot doit effectuer des mouvements pour le rendre plus attractif. Pour cela les mouvements qu'il effectue doivent être le plus naturels possible pour que ça se rapproche d'un comportement humain. Pour cela, le robot peut bouger sa tête à l'aide de trois moteurs combinés en un système appelé "orbita" ainsi que ses 2 antennes. Nous allons donc utiliser la combinaison des 2 pour effectuer les mouvements du robot. Comme dit dans le cahier des charges, nous avons utilisé l'API `reachy_sdk` pour mouvoir Reachy.

4.5.1 Faire différentes émotions

Pour que Reachy ait certaines réactions humaines il nous fallait définir des positions pour que le robot donne l'impression qu'il ressent des émotions. Dans le cahier des charges nous avions défini 6 positions décrites dans les annexes.

La première modification qui a été apportée est de ne pas définir une position fixe pour chaque émotion, mais de définir des angles fixes à partir de la position initiale. Cela permettra au robot de faire l'émotion en face de l'utilisateur peu importe sa position. Nous avons donc décidé de décrire les mouvements du robot dans un repère sphérique. De plus, nous avons défini les attributs `THETA` et `PHI` qui stocke les angles auxquels se trouve l'utilisateur et `TMP_THETA` et `TMP_PHI` qui contiennent la position actuelle du robot. Avec `THETA` correspondant aux angles de rotation autour de l'axe y et `PHI` autour de l'axe z représentés sur la figure 23. `THETA` ayant la valeur de 0 quand il a la tête totalement vers le haut et `PHI` ayant la valeur de 0 quand il regarde en face de lui.

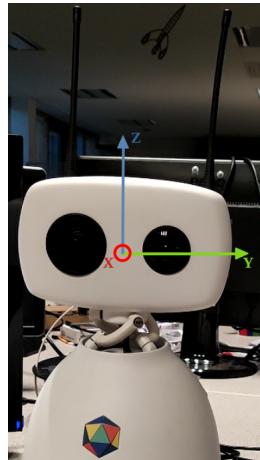


FIGURE 23 – Repère pour les mouvements de la tête

L'autre a été qu'au lieu de définir une durée que le robot doit mettre pour faire le mouvement, nous avons décidé de définir une vitesse. Cela permet à ce que peu importe la position d'où il se trouve et d'où il veut aller, il le fasse à la même vitesse. Cette modification permet de rendre le mouvement plus naturel. Avec uniquement une durée, il nous arrivait que le robot fasse un mouvement trop rapide et donc pas naturel, car il devait faire un petit mouvement pour aller de sa position initiale à sa position final ou au contraire un mouvement trop rapide pour faire un grand mouvement.

Les fonctions `listen`, `sad`, `happy`, `incentive`, `thinking` et `thanking` permettent respectivement à Reachy d'effectuer les émotions "écoute" 24a, "triste" 24b, "content" 24c, "incitation" 24d, "reflexion" 24e et "remerciement" 24f décrit dans le tableau ci-dessous 4.5.1.

Position	Angles tête	Vitesse tête	Angle antenne gauche	Angle antenne droite	Vitesse antennes
écoute	0, 0	0.15	0	0	/
triste	31.15, 0	0.13	140.0	-140.0	70
content	5.74, 0	0.15	± 20.0	± 20.0	300
incitation	-5.74, 0	0.1	35.0	-35.0	70
réflexion	-16.13, 16.7	0.21	-40.0	40.0	70
remerciement	5.74, 0/26.51, 0	0.15/0.35	-40.0	40.0	70

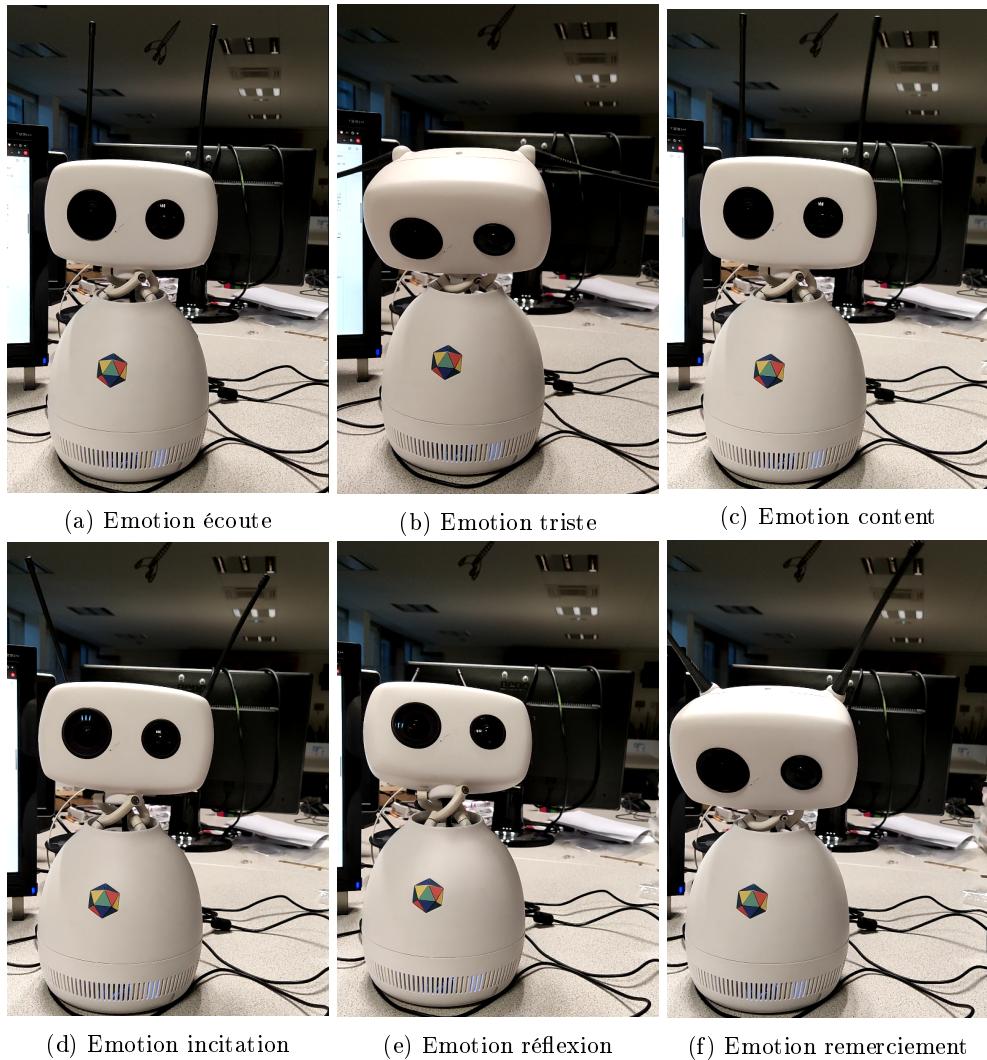


FIGURE 24 – Différentes émotions de Reatchy

Une fois que le robot a fini de faire son émotion, il faut qu'il retourne à sa position initiale face à l'utilisateur. Cela signifie que les angles doivent repérer TMP_THETA = THETA et TMP_PHI = PHI, c'est la fonction *move_back* qui permet de le faire.

Comme défini dans le cahier des charges, nous avons mis en place 6 émotions différentes que nous considérons comme principales. Pour pouvoir avoir encore plus d'interaction avec l'utilisateur, il pourrait être intéressant d'en définir de nouvelles afin qu'il ait un comportement plus proche d'un humain.

4.5.2 Suivi et cadrage des utilisateurs

Une fois que les émotions ont été définies, il était fixé, dans le cahier des charges, que le robot suive le visage, de la personne avec qui il interagit, pour rendre le robot plus interactif. Pour cela, il nous a fallu relier la partie capture d'image avec la partie mouvement.

La mise en relation des 2 parties, nous a dans un premier temps posé quelques difficultés, car les directions des axes pour le mouvement et ceux de la capture d'image n'étaient pas les mêmes. Une fois la fonction de conversion déterminée, nous avons pu voir le robot déplacer sa tête à l'endroit où la personne se trouve. Dans

un premier temps, pour effectuer ce mouvement nous avions utilisé la fonction `look_at` de l'api `reachy_sdk` comme pour faire les émotions, mais cela générait un problème assez important. Ce problème était que lorsque le robot tournait la tête sur un axe horizontal, il penchait la tête sur le côté. Ceci était gênant pour l'aspect naturel du mouvement, mais surtout pour la capture d'image. Cette dernière ne pouvait pas prendre en compte le fait que le robot penche la tête sur le côté, faisant que le calcul des angles était biaisé et par conséquent le robot, n'ayant plus le bon angle, se décalait au cours du temps. Il était donc impossible d'avoir un bon fonctionnement du suivi des visages.

Nous avons fini par comprendre que pour regarder un point dans l'espace le robot a plusieurs possibilités de le faire. La fonction `look_at` ne permet pas de spécifier la manière dont nous voulons qu'il regarde. En regardant l'ensemble de l'api `reachy_sdk`, nous avons remarqué que la fonction `inverse_kinematic` permet de choisir exactement la position de la tête du robot. La fonction `inverse_kinematic` prend en paramètres des quaternions pour pouvoir définir précisément la position des 3 moteurs déplaçant la tête de Reachy. Les quaternions sont des nombres complexes à 4 dimensions $Q = a + bi + cj + dk = [a \ b \ c \ d] = [\cos(\frac{\Theta}{2}) \ V_x * \sin(\frac{\Theta}{2}) \ V_y * \sin(\frac{\Theta}{2}) \ V_z * \sin(\frac{\Theta}{2})]$ avec V_x , V_y et V_z les coordonnées du vecteur dans le repère cartésien et Θ l'angle de rotation autour de ce vecteur comme l'illustre la figure 25. Il était plus simple pour nous de décrire et visualiser les mouvements avec les angles d'Euler, nous avons donc implémenté la fonction `_euler_to_quaternion` permettant de faire la conversion des angles d'Euler en quaternions. Une fois les valeurs du quaternions souhaités obtenus, nous déplaçons la tête dans la position voulue à l'aide de la fonction `update_position`. Le passage par la cinématique inverse nous a permis de résoudre le problème que nous avions rencontré.

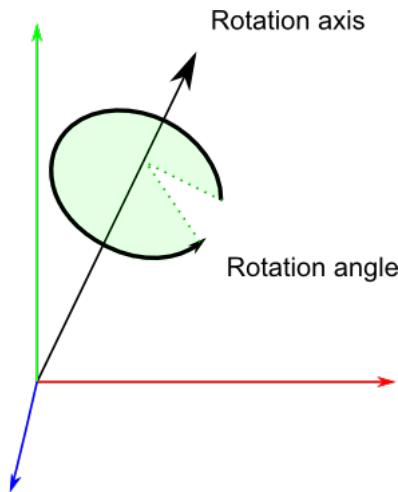


FIGURE 25 – Représentation quaternions

Reachy arrive donc bien à suivre le visage de l'utilisateur. Il lui manque cependant un peu de réactivité dans le suivi, mais cela reste tout de même acceptable.

Pour la prise de photo et surtout le cadrage de la photo, il est nécessaire que Reachy tourne sa tête vers la personne (ou le groupe de personnes) qui souhaite être prise en photo. Ayant déjà réussi le suivi de visage, le cadrage de la photo peut être considéré, du point de vue du mouvement, comme une partie de celui-là. C'est-à-dire qu'à partir des angles, permettant le cadrage, l'appel à la fonction `update_position` va orienter la tête de Reachy dans la bonne position pour prendre la photo comme s'il ne suivait qu'une unique fois un visage.

La partie cadrage ne nécessitant qu'un seul mouvement, il n'y a pas de problème de fluidité.

4.5.3 Ressemblance aux mouvements humain

Un point important du cahier des charges était que les mouvements devaient être les plus naturels possible. Donc une fois que nous sommes arrivés à ce que Reachy fasse les mouvements vers la direction que l'on souhaitait, il nous a fallu réfléchir ce qui faisait que son mouvement ne semblait pas naturel pour un être humain. Plusieurs aspects nous ont semblé importants de corriger pour qu'il se rapproche plus d'un comportement humain.

Tout d'abord contrairement à un humain qui est limité dans ces mouvements à cause de ces différentes articulations, le robot lui n'a pas de contrainte physique qui le limite dans ses mouvements. Donc dans un

premier temps pour ne pas endommager le câble qui est situé à l'arrière de sa tête, permettant la communication entre ses différents capteurs et son unité de traitement, nous ne lui demandions pas de tourner sa tête à plus de 180 degrés sur l'axe horizontal. Cela était tout de même risqué car rien, de manière physique ou logiciel, ne lui empêchait de faire un mauvais mouvement. De plus, pour qu'il se rapproche plus d'un comportement humain, il ne fallait pas simplement l'empêcher de faire un angle plus grand que 180 degrés, mais le limiter à l'angle maximal qu'un humain peut faire, que ce soit sur l'axe vertical ou horizontal. Nous avons donc regardé les angles maximums moyens qu'un être humain peut faire dans les différentes positions. Nous sommes arrivés aux valeurs suivantes :

- axe vertical vers le haut : 45 degrés
- axe vertical vers le bas : 40 degrés
- axe horizontal vers la gauche : 45 degrés
- axe horizontal vers la droite : 45 degrés

Une fois que nous avions déterminé ces valeurs, nous avons implémenté la fonction `__fit_angles` permettant de manière logicielle de limiter le mouvement de la tête du robot. Cette fonction est appelée à chaque fois avant de demander au robot de bouger. Elle va dans un premier temps regarder si l'angle duquel nous souhaitons bouger est bien dans l'intervalle, si c'est le cas, il ne fait rien sinon il modifie la valeur de l'angle duquel le robot va se déplacer vers la limite la plus proche. Cela signifie que si on demande de tourner la tête de 60 degrés vers la droite et de 46 degrés vers le haut, il tournera finalement de 45 degrés vers la droite et le 45 degrés vers le haut.

Ensuite, nous avons rencontré un problème par rapport à la vitesse de mouvement de la tête. Au début, nous avions défini une durée du mouvement peu importe la rotation qu'il devait effectuer. Cela faisait que pour certains mouvements il allait très doucement et que pour d'autres au contraire, il allait très vite. Cela n'est pas très naturel, car en principe un être humain se déplace à la même vitesse peu importe le mouvement qu'il effectue. Nous avons donc décidé de définir pour chaque mouvement et émotions une vitesse fixe. Les fonctions `look_at` et `goto` de l'api `reachy-sdk` prennent en paramètre la durée du mouvement et non pas la vitesse. Nous avons donc créé la fonction `__duration` qui calcule la durée que le mouvement va mettre en fonction de la vitesse souhaitée, de la position où il se trouve et de la position vers laquelle il se dirige.

Finalement, une fois les problèmes précédents résolus, pour rendre le mouvement encore plus naturel, nous avons essayé de fluidifier le mouvement. Nous avons évité d'enchaîner trop rapidement de petits mouvements pour que la tête de Reachy ne fasse pas des à-coups. Les mouvements que fait le robot sont donc fluides, mais il serait possible encore de les améliorer. C'est surtout le cas pour la partie suivi de visage qui en calculant le prochain endroit à atteindre avant d'avoir fini le mouvement en cours pourrait être moins saccadé. Cela permettrait d'éviter qu'il s'arrête pour repartir ainsi que de rendre le suivi plus précis. Cependant avec les fonctions utilisées de l'api `reachy-sdk`, il est obligatoire de finir le mouvement avant d'en faire un autre à la suite. Une amélioration qui serait possible serait d'utiliser une interpolation différente de celle linéaire pour que le mouvement soit légèrement plus naturel.

5 Mise en place et utilisation du robot / Manuel d'utilisation

Cette partie vise à expliquer comment faire fonctionner le robot sur les machines à états déjà implémentées du branchement du robot jusqu'à l'exécution de la `State_Machine`. Elle décrit également comment créer une nouvelle machine à états et l'exécuter.

5.1 Mise en place et matériel nécessaire

Le Reachy Mini est en fait un ordinateur sous Linux, de ce fait, la mise en place se fait comme celle d'un PC. Il est pourvu de 2 ports USB (voir (2) sur la Figure 26), 2 ports HDMI (voir (3) sur la Figure 26) et un port d'alimentation (voir (1) sur la Figure 26). Il est donc nécessaire d'avoir un clavier et une souris pour les brancher aux 2 ports USB afin de naviguer dans le Reachy comme avec un PC. Le microphone ne fonctionnant pas très bien, il y a la possibilité de brancher un microphone externe via un casque-micro ou bien un micro seul. Pour cela il faut se munir d'un hub USB et le brancher à l'un des 2 ports USB puis y brancher le clavier, la souris et le micro. Ensuite pour ce qui est du visuel il faut brancher un écran sur l'un des ports HDMI. Durant le projet nous n'avions que des écrans avec entrée VGA, c'est pourquoi nous avons utilisé un adaptateur HDMI vers VGA, il est donc possible d'utiliser un adaptateur en fonction des besoins si vous ne disposez pas d'un écran avec entrée HDMI. Enfin, il suffit de brancher le câble d'alimentation du Reachy et de l'écran et la partie connectique est prête.

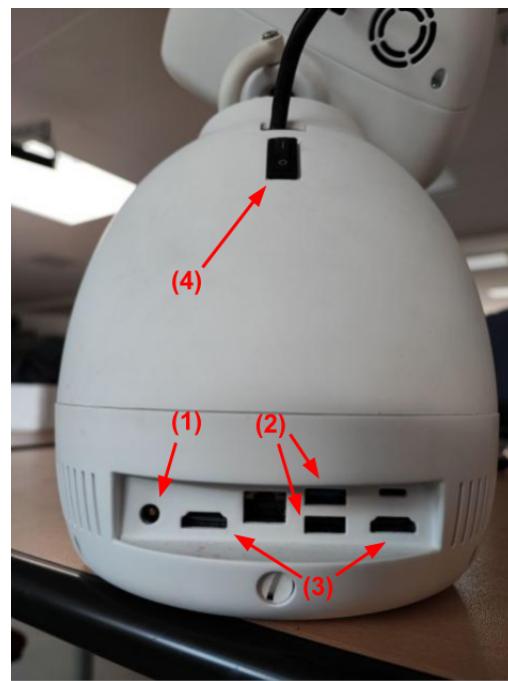


FIGURE 26 – Différentes connectiques du Reachy et bouton d'activation des moteurs

Maintenant que le Reachy est branché, il suffit d'appuyer sur le bouton d'allumage de celui-ci qui est placé sur son torse (voir sur la Figure 27). Ensuite l'ordinateur s'allume et nous pouvons naviguer comme sur tout autre ordinateur Linux via l'écran, le clavier et la souris.



FIGURE 27 – Position du bouton d'allumage du Reachy

5.2 Initialisation du robot

Tout d'abord il faut se placer dans le dossier du projet dont le chemin est `home/gitdir/PFA-Reachy-mini`, par la suite, toutes les commandes sont à effectuer depuis la racine du projet.

Avant de lancer une machine à états, il faut installer toutes les librairies et modules nécessaires à son utilisation. Pour cela, il faut exécuter la commande :

```
$ make install
```

Également, il est nécessaire de rafraîchir les serveurs du robot avec la commande :

```
$ make reload-server
```

En effet, cela permet d'éviter quelques soucis de connexion.

Ensuite, il est nécessaire d'effectuer le focus de la caméra du Reachy. Pour cela il faut tout d'abord activer les moteurs du robot en appuyant sur l'interrupteur placé sur le dos du robot (cf (4) Figure 26). Ensuite, il faut lancer le programme Python dédié au focus. Celui-ci peut être exécuté via la commande :

```
$ make initiate-camera
```

Une fenêtre va s'afficher avec le retour vidéo du Reachy. Il vous faudra alors placer votre main proche de la caméra pour que le Reachy fasse le focus sur elle (cf Figure 28a). Puis l'éloigner petit à petit en lui laissant le temps de faire les focus successifs (cf Figures 28b). Enfin, lorsque l'image semble nette (cf Figure 28c) il faut appuyer sur la touche Q de votre clavier pour fixer le focus. Il est tout à fait possible de refaire le focus s'il ne vous convient pas en refaisant la même manipulation.

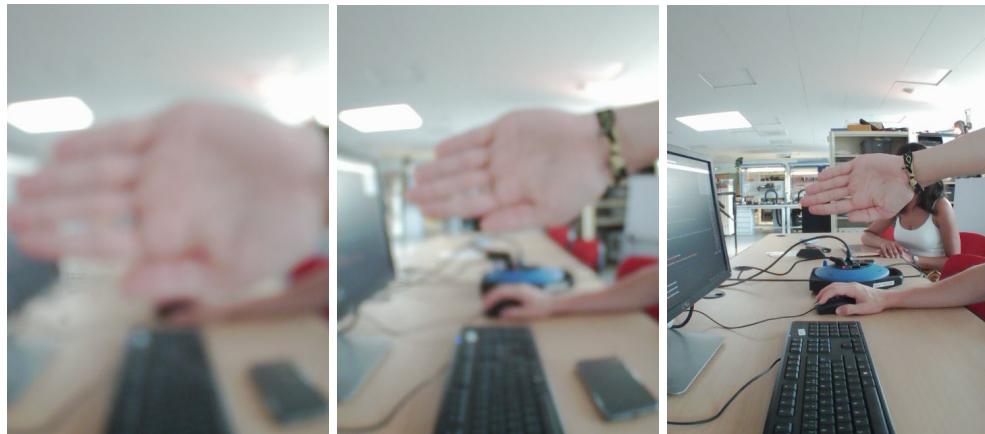


FIGURE 28 – Suite d'images expliquant les étapes d'initialisation de la caméra du Reachy

Cela termine la partie mise en place du robot.

5.3 Utilisation du robot

Afin de décider des fonctionnalités du robot, il faut se servir de la partie sur la machine à états (cf 3.3). Il faut donc tout d'abord créer un fichier `.json` comme l'annexe 9.2 en spécifiant les états, les transitions, les actions et prédicts associés et en le plaçant dans le dossier `/PFA-Reachy-mini/assets/json`. Ensuite, il faut créer un fichier considéré comme `module` de la machine à états c'est-à-dire un fichier avec l'implémentation de l'ensemble des fonctions demandées par la `State_Machine`. Celui-ci devra être placé dans le dossier `PFA-Reachy-mini/src/state_machine`. Avec le fichier `.json` et le fichier `module`, il est ensuite possible de lancer le Reachy sur cette machine à états. Pour cela il faut rajouter une règle dans le `Makefile` en copiant par exemple la règle `reachy-final` et en remplaçant son nom par le nom voulu, la description de la machine par la description souhaitée et le fichier `state_machine_final.json` par le nom de notre fichier `.json`.

Cette partie présente comment lancer une machine à états quelconque et présente celle déjà présente dans le robot et comment faire exécuter par Reachy.

5.3.1 Machine à états quelconque

Pour exécuter une `State_Machine` sans passer par le `Makefile` il faut se rendre dans le dossier `/PFA-Reachy-mini/src/state_machine`. Ensuite il faut lancer un programme Python en lui spécifiant le chemin relatif vers le fichier `.json` de la machine. Pour cela il suffit d'exécuter la commande suivante :

```
python3 reachy_state_machine_argv.py <chemin relatif vers le .json>
```

Ceci est la méthode pour lancer n'importe quelle machine à états à partir de son fichier `.json`. Nous allons maintenant donner les méthodes pour lancer les 2 machines principales du projet.

5.3.2 Machine à états "Only Aruco"

La première est la machine à états qui n'utilise que les codes Aruco. Il faut donc se munir des 16 premiers codes Aruco afin de pouvoir utiliser toutes ses fonctionnalités. Il est possible d'avoir une conversation basique avec le robot, de prendre une photo de groupe, une photo simple, d'effectuer un échange de visage et une photo en noir et blanc. Pour exécuter cette machine à états et donc cette version du robot, la commande à utiliser est :

```
make reachy-only-aruco.
```

5.3.3 Machine à états "finale"

Il s'agit de la machine à états la plus aboutie. Le robot attend tout d'abord d'entendre "Hey Reachy" avant de s'activer. Puis écoute la personne. Il est capable de reconnaître différents mots-clés et dans le cas où il ne reconnaît rien il fait appel à l'IA GPT-3. Il est également capable des mêmes capacités en termes de prise de photo que le "Only Aruco". Pour exécuter cette machine à états et donc cette version du robot, la commande à utiliser est :

```
make reachy-final.
```

5.4 Autres programmes utiles

Cette partie décrit les programmes annexes pouvant être exécutés en parallèle du Reachy afin d'effectuer la gestion des images prises par le robot.

5.4.1 Programme d'affichage des photos

Lors de la prise de photos, celles-ci sont stockées dans le Reachy. Il est possible de les afficher en temps réel en lançant le programme dédié en parallèle de la `State_Machine`. Pour cela il suffit de lancer la commande suivante :

```
$ make show-last-img
```

Cela va afficher une fenêtre avec la dernière photo prise, cela ne s'actualise pas automatiquement si le Reachy prend une photo, il faut presser la barre espace afin de rafraîchir la fenêtre. Ce programme a été développé pour permettre l'affichage des photos en plein écran, ainsi il est préférable d'utiliser un deuxième écran afin de pouvoir afficher les images et manipuler le robot en même temps. Cependant, il est tout à fait possible d'utiliser ALT + TAB pour basculer entre les deux.

5.4.2 Programme de suppression de photos

La mémoire du Reachy est bien entendu limitée, c'est pourquoi nous avons mis en place un programme de suppression de photos. Celui-ci peut se lancer en parallèle d'une machine à états et de l'affichage des photos. Son rôle est de supprimer les photos les plus anciennes si le Reachy a déjà enregistré N photos. Le programme permet donc la suppression de P photos à partir du moment où il y a N photos ou plus dans le répertoire de stockage des photos. N et P sont donc les arguments à donner en ligne de commande. Ainsi, pour le lancer il faut exécuter la commande suivante :

```
$ make delete-images NB_PHOTO= <N> NB_TO_DELETE= <P>
```

Par défaut, le nombre d'images à stocker avant suppression est fixé à 100. Et dès que cette valeur est atteinte, 2 images sont supprimées.

6 Les limites du robot et les améliorations possibles du projet

Lors de ce projet, nous avons manipulé Reachy sous différents aspects. Ainsi, certains aspects positifs et négatifs ont été repérés et cette partie va permettre de lister ces caractéristiques afin d'offrir à Pollen Robotics des voies d'amélioration possible du robot Reachy Mini, et également afin d'expliquer pourquoi certaines parties du projet ont été plus ou moins efficaces.

6.1 Les qualités et défauts de Reachy Mini

Qualités

Tout d'abord, le robot présente de nombreux avantages qui facilitent sa manipulation. La grande modularité de ses fonctions était un atout très important pour permettre une avancée rapide de notre projet. En effet, comme nous manipulions différentes parties du robot, il était très utile que la synthèse vocale, les mouvements et les caméras du robot soient manipulables de façon modulaire.

Également, la présence de nombreuses fonctions bas niveau est un grand atout. Par exemple, pour les mouvements, le fait de pouvoir contrôler la tête du robot avec les méthodes `look_at` et `goto` permet de contrôler sa tête de deux façons, la première concernant le regard du robot, et la deuxième utilisant directement les angles des joints liants la tête au corps. Ainsi, la mise en place des émotions, et des mouvements et leur différenciation a été facilitée par l'utilisation de ces méthodes.

Défauts

Cependant, le robot présente tout de même quelques défauts. Le premier défaut que nous avons constaté concerne les mouvements. Il semblerait que le robot garde en mémoire (en cache) le mouvement qu'il est en train de réaliser. Nous avons eu plusieurs fois des segmentation fault lors de ce projet et si nous rallumions le robot, alors la première chose qu'il cherchait à faire lors de l'appel à une fonction de mouvement, était de faire le dernier mouvement qu'il avait essayé de faire et qui n'était pas encore terminé. Ainsi, nous avons plusieurs soucis liés à cela, sans réellement trouver la cause de ce redémarrage. Cependant, après avoir mis des vérifications sur les angles envoyés aux fonctions de mouvement, nous n'avons plus eu ce problème, car le robot ne faisait que des mouvements "valides" et donc ne crashait plus de cette façon.

Un autre défaut, plus important cette fois-ci, concerne le micro du robot. Nous avions besoin de ce micro pour la reconnaissance vocale. Cependant, bien qu'il capte parfaitement la direction du son (micro multi-directionnel), il n'était pas en mesure de capturer suffisamment le son pour que le robot comprenne ce qui était dit. Même une fois la réduction de bruit mise en place, le robot captait le son, dans la bonne direction, mais il ne captait pas les paroles réellement prononcées.

Nous avons essayé ce micro de façon externe au robot avec une carte audio supplémentaire. Cependant, nous avons obtenu les mêmes problèmes. Enfin, en ajoutant un micro externe au robot, tel qu'un casque-micro, alors le son est parfaitement capté. Le soucis provient donc bien du micro utilisé dans le robot.

Ainsi, il a été impossible d'utiliser le micro directionnel permettant au robot de s'orienter vers le son.

Enfin, un autre défaut du robot a été l'utilisation de ses caméras. Il se trouve que le focus était compliqué à réaliser car les caméras fixaient automatiquement les micro-poussières présentes à l'intérieur de la lentille. Pour contrer cela, nous avons choisi de réaliser le focus une unique fois à l'initialisation du robot, puis de ne plus y toucher. On pouvait ainsi avoir un réel aperçu du focus et le calibrer comme nous le souhaitions. De plus, une fois le focus calibré, il permet la netteté d'une grande zone du champ de vision du robot, il n'est donc pas nécessaire de le refaire à chaque fois.

6.2 Améliorations possibles de notre projet

Envoi des photos

En ce moment, toutes les photos prises par Reachy sont sauvegardées localement. Une façon plus sophistiquée de le faire serait de faire une interface qui affiche un champs de texte pour que chaque utilisateur puisse rentrer son adresse e-mail, et par la suite on pourra la lui envoyer.

Utilisation des deux caméras

Pour des soucis de simplification, nous avons fait le choix de n'utiliser que la caméra droite du robot. En effet, elle procure un angle suffisamment grand pour être utilisée toute seule. Une amélioration possible du projet serait d'utiliser les deux caméras et de les exploiter de telle sorte qu'elles ne forment qu'une seule caméra plus performante.

Ajouts d'émotions

Les émotions que Reachy peut exprimer jusqu'à présent sont limitées à six émotions : `écoute`, `triste`,

content, incitation, réflexion, remerciement. Ces émotions ne peuvent pas être exprimées au milieu d'une conversation avancée. Alors deux améliorations sont envisagées dans ce cas : d'abord, nous souhaitons ajouter plus d'émotions comme par exemple **ennuyé** ou **surpris**, et nous voulons intégrer un algorithme d'apprentissage automatique qui analyse, au fur et à mesure, toutes les phrases de l'utilisateur, et ce au milieu de la conversation avancée, et qui donne trois réponses : **positif, négative** ou **neutre**. Par la suite, nous associons le mouvement associé à chacune de ces réponses. Ceci peut être également fait avec l'API **OpenAI**.

Parallélisation des fonctions de détection

Actuellement, nous avons deux machines à états différentes pour interagir avec le robot soit avec des codes Aruco, soit avec la voix. Une amélioration sur laquelle nous avons commencé à travailler est la parallélisation des processus. En effet, il serait très utile que le robot puisse, tout en écoutant ce qu'il se passe autour de lui, observer les choses et donc détecter des codes aruco. De même, cela permettrait de réaliser de nombreuses actions en parallèles sans forcément utiliser les fonctions asynchrones de l'API **reachy-sdk**.

Nous avons commencé à implémenter cette amélioration, cependant nous avons rencontré des soucis avec **PyAudio** qui n'aimait pas être exécuté en parallèle à un autre programme.

Nous avons donc choisi de nous concentrer sur les autres fonctionnalités quitte à faire deux machines à états différentes plutôt que s'obstiner à faire marcher une unique machine à états qui n'aurait pas eu toutes les fonctionnalités.

Utilisation du micro directionnel en plus d'un autre micro

Une amélioration possible serait de combiner la fonctionnalité du micro associé au robot qui détecte la direction du son avec l'autre micro que nous avons ajouté qui détecte bien le son et les paroles afin d'avoir un bon rendement au niveau de la détection du son.

7 Évènements de présentation du projet

L'un des objectifs finaux du projet était de pouvoir présenter Reachy lors de différents événements. Cela permettant aux différents participants de pouvoir se prendre en photo sur demande.

7.1 Soirée Partenaires

Le premier événement auquel il a vraiment participé a donc été la soirée partenaire de l'ENSEIRB-MATMECA. Cela nous a permis de présenter notre travail à certains des élèves ainsi qu'aux partenaires de l'école.

En prenant en compte le niveau sonore du bruit dans la pièce de l'événement, nous avons préféré ne pas utiliser la partie reconnaissance vocale du robot. Pour cela nous avons donc mis en place une machine à états où la communication des ordres ne s'effectue qu'à l'aide des tags aruco.

Même si la commande vocale ne fonctionnait pas, les tags arucos ont permis d'effectuer la plupart des fonctionnalités du robot. Reachy effectuait les mouvements et émotions correspondants aux actions et aux commandes demandées. Par exemple lorsque nous lui montrions le tag correspondant à "tu es mignon", le robot faisait l'émotion "content". Reachy pouvait aussi tourner la tête pour détecter et suivre la personne face à lui. L'utilisateur pouvait ainsi lui demander de prendre une photo, seul ou en groupe. Reachy cadrerait donc la photo en bougeant la tête avant de prendre la photo et de l'afficher sur l'écran. L'utilisateur pouvait aussi demander au robot de lui raconter une histoire. A l'aide d'**openIA** et du contexte de l'événement donné en paramètre, Reachy pouvait donc raconter des histoires en relation avec la soirée partenaire.

Cet événement a donc été très intéressant pour montrer les différentes fonctionnalités du robot que nous avons implémentées ainsi que de prendre en compte les remarques des utilisateurs afin de l'améliorer davantage.

7.2 100 ans de l'ENSEIRB

Les 100 ans de l'ENSEIRB était l'événement majeur cité dans le cahier des charges. Le robot devait être fonctionnel et être présent pour cet événement. Cependant, à cause des circonstances actuelles, les 100 ans de l'ENSEIRB n'ont pas eu lieu à la date initiale. Il est cependant prévu que cet événement soit reporté à début septembre de cette année. Si cela est possible, nous essayerons d'y participer afin de montrer le fonctionnement et l'ensemble des possibilités que Reachy offre en espérant que cette fois-ci nous pourrons utiliser la partie reconnaissance vocale malgré le monde et le bruit qu'il y aura.

8 Conclusion

Le Projet au Fil de l'Année a été un projet intense dont le management n'a pas été très facile à gérer dû au grand nombre de personnes dans le groupe. Ainsi, des rôles se sont naturellement formés au sein du groupe pour assurer une bonne coopération. Par exemple, Lucas gérait la communication avec le client et l'encadrant, Clara mettait en commun les différents modules du code, et chacun des autres membres était concentré sur l'élaboration de sa partie et de sa bonne réalisation.

De plus, nous n'avions pas de créneaux réservés à la réalisation du projet dans nos emplois du temps respectifs. Mais nous avions la contrainte de travailler avec le robot pour certaines parties comme celle des mouvements. Ainsi, il a fallu trouver des créneaux communs pour travailler. En général, nous nous retrouvions le jeudi matin, et en début d'après-midi pour travailler sur le robot (les parties non faisables à la maison notamment) puis nous enchaînions par une réunion avec l'encadrant ou le client.

M. ROLLET était notre encadrant, bien que M. MORANDAT ait également participé à la gestion du projet. Ce regard nouveau sur la réalisation, nous a permis d'obtenir un nouvel avis, et a été un moteur pour la suite du développement du projet. En effet, les bases ayant été fixées par M. ROLLET et le groupe de PFA et le développement de ces-dernières ayant déjà été commencé, M. MORANDAT a pu avoir un regard extérieur et soulever des contraintes et améliorations auxquelles nous n'avions pas encore réfléchis. Ce qui constituait ainsi une petite perte de temps car il fallait revoir quelques parties de l'implémentation, a finalement permis de gagner en rapidité sur la suite du projet (par exemple avec la nouvelle façon de créer des machines à états à l'aide des json).

Lors de l'intégralité du projet, nous pouvions entrer en contact avec des membres de Pollen Robotics. En effet, nous avions été ajoutés au serveur Discord de l'entreprise par M. N'GUYEN. Cela nous a permis notamment d'obtenir des informations supplémentaires sur certaines documentations que nous ne trouvions pas sur le Github de Pollen Robotics. Cette aide a donc été précieuse afin de poser nos questions directement aux personnes connaissant le robot et ses fonctionnalités.

Ainsi, le PFA sur le robot Reachy a été un projet innovant pour l'ensemble du groupe. Nous avons apprécié travailler sur différents aspects (mouvement, caméra, audio, ...) et sur un projet concret. Voir concrètement le robot bouger ou parler à la suite de l'implémentation de certaines fonctions était très motivant pour la suite du projet.

Remerciements

L'ensemble du groupe tient à remercier M. ROLLET Antoine, encadrant du projet, ayant suivi le groupe tout au long du projet, même à distance.

Nos remerciements vont également à M. MORANDAT Floréal, qui a permis la continuité pédagogique lors du remplacement temporaire de M. ROLLET. Son regard extérieur a permis de faire émerger de nouvelles idées qui ont favorisé la réalisation du projet.

Merci à M. JANIN David pour la gestion des PFA, la proposition des sujets et les informations diverses qu'il a pu partager.

Nous tenons également à remercier M. N'GUYEN Steve pour avoir proposé ce projet en tant que PFA, et nous avoir fait confiance dans la réalisation de ce dernier. Son aide a également été précieuse pour comprendre le robot et son API.

Nous remercions aussi POLLEN ROBOTICS et l'ensemble de ses membres pour leur réactivité sur le serveur Discord, mais également pour le prêt du robot sans qui le projet n'aurait pas eu lieu.

Bibliographie

Figure 25 : image représentation des quaternions
<http://www.opengl-tutorial.org/fr/intermediate-tutorials/tutorial-17-quaternions/>

Figure 17, 18 et 19 : face swapping
<https://pysource.com/2019/05/28/face-swapping-explained-in-8-steps-opencv-with-python/>

9 Annexes

9.1 Cahier des charges

FILIÈRE INFORMATIQUE - SEMESTRE 8

Cahier des Charges

Application d'assistant vocal sur Reachy Mini



Auteurs :

RAÏS Sylvain
ZIZOUAN Widad
DIEUDONNÉ Clara
MARAIS Lucas
LAMHAMDI Aymane
ELFANI Hamza
BOUDEAU Benjamin

Encadrant :

ROLLET Antoine

Client :

N'GUYEN Steve

Table des matières

1	Introduction	2
2	Besoins fonctionnels et non fonctionnels	3
2.1	Besoins fonctionnels	3
2.2	Besoins non fonctionnels	4
3	Partie 1 : Les 100 ans de l'ENSEIRB	5
3.1	Diagramme d'états	5
3.2	Actions de Reachy selon les états et transitions	6
3.2.1	États	6
3.2.2	Transitions	7
3.3	Débuter une interaction avec le robot	8
3.4	Gestion des ordres	8
3.5	Prise de photos	9
3.6	Gestion des photos	10
3.6.1	Sauvegarde de la photo	10
3.6.2	Affichage de la photo sur un écran	10
3.7	Conversation	10
3.8	Fin d'interaction	10
4	Partie 2 : Après les 100 ans	11
4.1	Diagramme d'états	11
4.2	Filtres sur les photos	12
4.3	Recherche Internet	12
4.4	Conversations plus interactives	12
5	Deux cas d'utilisation	14
5.1	Interaction directe	14
5.2	Interaction par détection de visage	15
6	Tâches et livrables	16
6.1	Les étapes de réalisation techniques	16
6.1.1	La reconnaissance faciale	16
6.1.2	Mouvement du robot	16
6.1.3	Gestion des ordres	16
6.1.4	Synthèse vocale	17
6.1.5	Recherche Internet	17
6.2	Diagrammes de Gantt	17
6.3	Livrables	18
6.3.1	"Hello World" [07/02/22]	18
6.3.2	Full Interaction [16/02/22]	18
6.3.3	Le robot communique avec émotions [25/02/22]	18
6.3.4	Le robot sait prendre des photos [11/03/22]	18
6.3.5	Le robot communique et prend des photos [18/03/22]	18
6.3.6	100 ans de l'ENSEIRB [01/04/22]	18
6.3.7	Système de conversation amélioré [23/04/22]	18
6.3.8	Recherche internet pour des conditions de surf [30/04/22]	19
6.3.9	Full version [07/05/22]	19
7	Webographie	20
8	Annexe	21
8.1	Définition des positions	21
8.2	Ensembles de mots clé	21
8.3	Couple pour la conversation basique	23
8.4	Ensembles de phrases/mots clés pour la conversation avancée	24

1 Introduction

Le Projet au fil de l'année (PFA) est un projet regroupant 7 élèves ingénieurs de l'ENSEIRB-MATMECA, qui devront réaliser un projet pour un client jusqu'au 15 mai 2022. Ce PFA concerne le robot Reachy Mini et le client, N'GUYEN Steve, est un membre actif de l'entreprise Pollen Robotics, entreprise ayant développé le robot.

Fondée en 2016 par d'anciens chercheurs, Pollen Robotics est une entreprise qui rassemble des personnes talentueuses et indépendantes dans le but de fournir des produits et des applications accessibles et open source et d'amener l'IA et la robotique dans notre vie quotidienne. L'entreprise participe donc à l'évolution de l'IA dans la vie quotidienne, et expose son robot principal Reachy sur de nombreux évènements autour de l'informatique, la robotique et l'intelligence artificielle.

Le robot Reachy Mini est l'une de leur création, aux côtés de son grand frère Reachy (robot articulé possédant des bras). Il possède une tête articulée et des antennes, le tout associé à un corps statique. Ses yeux sont formés par deux caméras, et il possède un microphone directionnel et des hauts parleurs. Fonctionnant grâce à un PC embarqué Intel NUC et un Google Coral, le robot possède tous les outils pour interagir avec le monde qui l'entoure.

Le travail demandé par le client est de programmer Reachy afin qu'il se rapproche d'un assistant vocal de type Amazon Echo, ou Google Home. La reconnaissance vocale, le traitement d'image, la synthèse vocale et les mouvements du robots seront donc à implémenter.

Le premier objectif est le bon fonctionnement du robot lors des 100 ans de l'ENSEIRB le 8 avril 2022. Le robot devra pouvoir interagir de façon simple avec les utilisateurs, ainsi que les prendre en photo selon leur volonté (photos simples, photos de groupes, ...).

Suite à cet événement, dans le prolongement du projet, les interactions seront développées de façon plus approfondie pour se rapprocher d'un réel assistant vocal avec notamment la recherche sur internet et des discussions plus naturelles.

Le cahier des charges se décompose en plusieurs parties, la première présentera les besoins fonctionnels, non fonctionnels et contraintes liés au projet. Ensuite nous diviserons ce projet en deux parties : le développement des fonctionnalités avant puis après les 100 ans de l'ENSEIRB. Enfin, nous développerons deux cas d'utilisation afin d'illustrer les capacités du robot, puis nous annoncerons les différents livrables qui seront rendus lors de la phase de construction du projet.



FIGURE 1 – Logo de Pollen robotics

2 Besoins fonctionnels et non fonctionnels

Afin de mener à bien ce projet, nous avons identifié différents besoins que l'on peut répartir en deux catégories, les besoins fonctionnels qui correspondent aux besoins de services (fonctionnalités du robot accessibles et visibles à l'utilisateur), et les besoins non fonctionnels qui correspondent aux autres besoins, c'est à dire aux besoins techniques.

2.1 Besoins fonctionnels

Les besoins fonctionnels nécessaires à ce projet sont répartis de la manière suivante :

La reconnaissance faciale :

- Identification de visage
- Suivi de visage
- Cadrage pour prise de photos
- Échange de visages
- Mise en noir et blanc
- Similarité de visages

Gestion des données :

- Sauvegarde de la photo dans le disque dur interne
- Gérer l'espace de stockage en cas d'incapacité à enregistrer une autre photo
- Afficher la photo sur un écran connecté au robot

Mouvement du robot :

- Mouvement de tête et des antennes séparément
- Simuler des émotions
- Mouvement permettant de suivre les personnes
- Mouvement pour se tourner vers le son

Gestion des ordres :

- Transcription de la voix en texte
- Atténuation du bruit extérieur moyen et fort
- Mise en place d'un système de conversation
- Gestion des marqueurs ArUco

Synthèse vocale :

- Conversion d'un texte en son sortant du robot
- Émettre de petits sons transcrivant une "émotions"

Recherche Internet :

- Recherche des informations nécessaires
- Extraction des informations nécessaires
- Énoncer les informations

2.2 Besoins non fonctionnels

Pour ce qui est des besoins non fonctionnels, nous avons :

Les API utilisées pour les différentes parties du projet :

- OpenCV (reconnaissance faciale)
- SpeechRecognition (reconnaissance vocale)
- PyAudio (reconnaissance vocale)
- pyttsx3 (synthèse vocale)
- eSpeak (synthèse vocale)
- wave (gestion des fichiers audio)
- pydub (reconnaissance vocale)
- reachy-sdk (mouvement du robot + accès aux caméras)
- webbrowser et request (recherche internet)

Les périphériques nécessaires au bon fonctionnement du robot :

- Écran
- Clavier
- Souris

Contraintes :

- Le robot possède une tête articulée lui permettant d'effectuer ses mouvements. Cependant, cette tête peut rapidement chauffer et se mettre en "sécurité" en arrêtant ses moteurs, le robot ne sera alors plus fonctionnel. Il est donc nécessaire que le robot fasse des mouvements essentiels uniquement, et de limiter leur envergure, afin de limiter la hausse de la température.
- La reconnaissance vocale nécessite une connexion internet pour fonctionner, cela fait donc partie des contraintes. Néanmoins l'un des objectifs étant la "Recherche Internet" le robot devra forcément être doté d'une connexion internet quelle que soit l'API utilisée.

3 Partie 1 : Les 100 ans de l'ENSEIRB

Cette partie développe les besoins explicités en section 2, en détaillant les sous-besoins, pour l'exposition du robot aux 100 ans de l'ENSEIRB-MATMECA le 8 avril 2022. Le robot peut être décomposé comme une machine à états où il réalise des actions en fonction des états dans lesquels il se trouve et des transitions empruntées. Nous détaillerons dans cette partie le diagramme de ces états. Ensuite, nous expliquerons les actions réalisées par Reachy sur les états puis sur les transitions.

3.1 Diagramme d'états

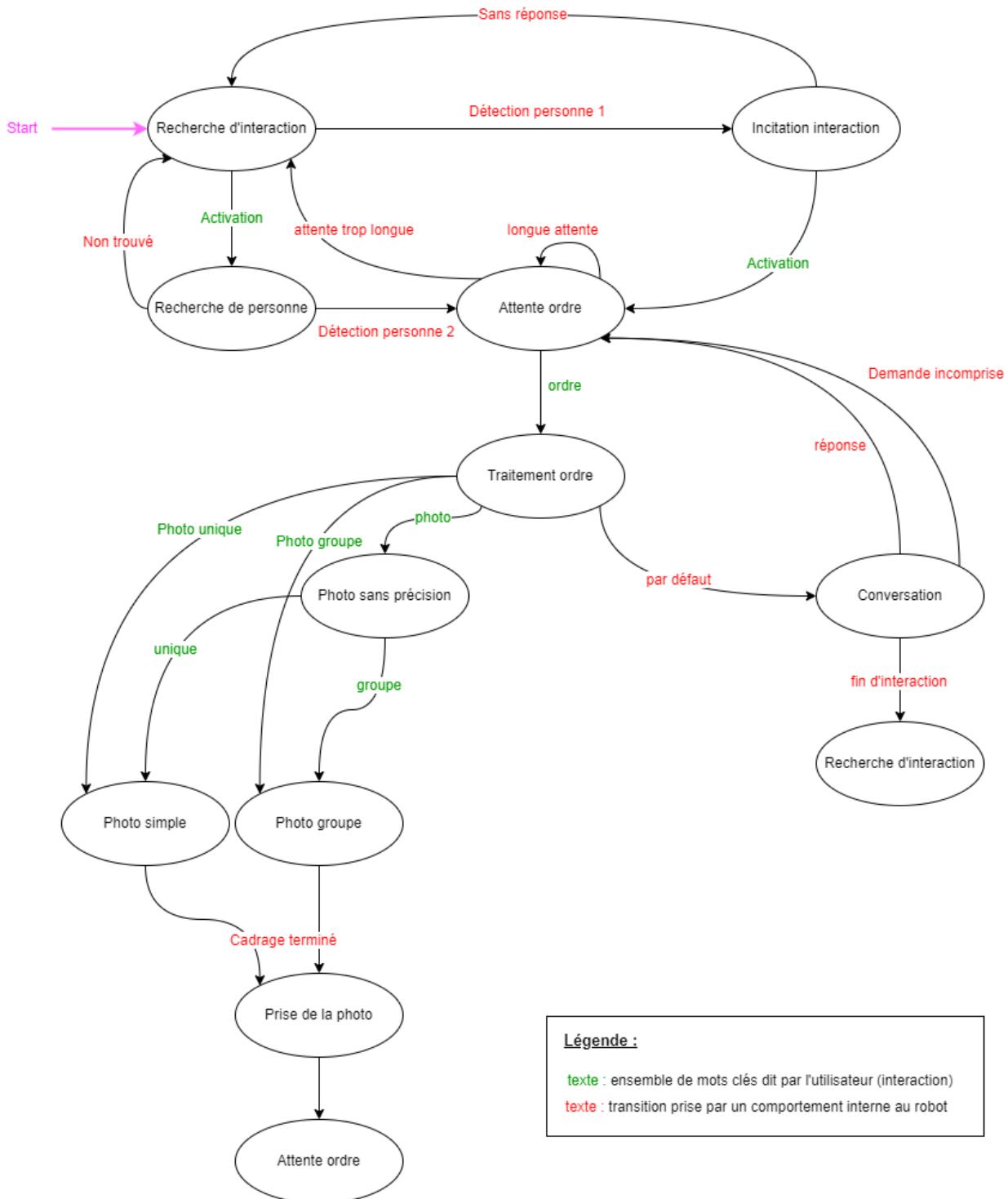


FIGURE 2 – Diagramme d'états pour les 100 ans de l'ENSEIRB

3.2 Actions de Reachy selon les états et transitions

Le diagramme est composé de deux parties : les états et les transitions. Sur chaque état et chaque transition, le robot réalisera des actions (mouvement, parole, ...) qui seront détaillées, d'abord pour les états, puis pour les transitions.

Chaque mouvement de la tête et des antennes du robot est mentionné par un *nom*, le détail technique des valeurs des positions est donné en **Annexe**, section 8.1, pour chacun de ses mouvements.

3.2.1 États

État "Recherche d'interaction" :

- Synthèse vocale : Fait des petits bruits toutes les 30 secondes
- Mouvement : Tourne la tête pour suivre la personne la plus proche du regard sauf toutes les 3 minutes où pendant 30s le robot fera le mouvement *triste*

État "Incitation interaction" :

- Synthèse vocale : Proposer une seule fois de dire "Hey Reachy" pour commencer l'interaction
- Mouvement : Fait le mouvement *incitation*

État "Attente ordre" :

- Mouvement : Fait le mouvement *écoute*

État "Traitement ordre" :

- Mouvement : Fait le mouvement *réflexion*

État "Recherche de personne" :

- Mouvement : Parcours de la zone possible avec recherche de visage (Cf OpenCV et direction sonore pour assister)

Etat "Photo sans précision" :

- Synthèse vocale : Demander "Vous préférez faire une photo simple ou une photo de groupe?"

Etat "Photo simple" :

- Mouvement : Cadrage selon OpenCV

Etat "Photo groupe" :

- Mouvement : Cadrage selon OpenCV

Etat "Conversation" :

- Synthèse vocale/Reconnaissance vocale : Si la phrase en entrée peut être interprétée, renvoie une réponse en fonction de la phrase en entrée (couples d'entrées / sorties), sinon (si la phrase d'entrée ne correspond à rien) dit "je n'ai pas compris votre demande"
- Mouvement : Fait le mouvement *réflexion*

Etat Prise de photo :

- Synthèse vocale : Dit "je prends la photo dans 5 4 3 2 1, dites Cheeeese"

3.2.2 Transitions

Transition "Détection personne 1" :

- Synthèse vocale : Après avoir trouvé, dit "Bonjour"
- Mouvement : Fait le mouvement *content*

Transition "Sans réponse" :

- Mouvement : Fait le mouvement *triste*

Transitions "Activation" :

- Synthèse vocale : Reachy répond "Hey, je vous écoute!" à la fin de la transition
- Mouvement : Fait le mouvement *écoute*

Transition "Longue attente" :

- Synthèse vocale : Dit "Je vous écoute, si vous voulez connaître l'ensemble de mes capacités demandez moi en disant "Que sais-tu faire ?""
- Mouvement : Fait le mouvement *incitation*

Transition "Attente trop longue" :

- Mouvement : Fait le mouvement *triste*

Transition "Non trouvé" :

- Mouvement : Fait le mouvement *triste*

Transition "Détection personne 2" :

- Synthèse vocale : Dit "Oh je vous ai trouvé!"
- Mouvement : Fait le mouvement *écoute*

Transition "par défaut" :

- Mouvement : Fait le mouvement *écoute* ou si l'utilisateur dit un mot triste ou gentil :
 - Mot triste → mouvement *triste*
 - Mot gentil → mouvement *content*

Transition vers "photo unique", "photo groupe", "unique" et "groupe" :

- Mouvement : Fait le mouvement *écoute*

Transition "réponse" :

- None

Transition "demande incomprise" :

- Mouvement : Fait le mouvement *incitation*

Transition "Fin d'interaction" :

- Synthèse vocale Dit "Au revoir, c'était un plaisir"
- Mouvement : Fait le mouvement *remerciement*

3.3 Débuter une interaction avec le robot

Avant toute interaction, le robot émettra de légers bruits (Coucou – hmmmm – ehooo – Ayee Ayee.) pour capter l'attention des individus à proximité.

Ensuite, deux types de début d'interaction seront développés sur le robot selon si le robot a détecté l'utilisateur avant qu'il prononce "Hey Reachy" ou non.

- Dans le cas où un utilisateur prononce "Hey Reachy" avant que le robot ne le détecte. La tête se tournera vers la zone où le micro multi-directionnel identifie la voix et la reconnaissance faciale permettra d'effectuer une recherche locale d'un visage estimé "assez proche" pour être l'interlocuteur.

Une fois la personne détectée, Reachy lèvera les antennes vers le haut et regardera la personne pour lui faire comprendre qu'il l'a bien vue (mouvement *écoute*) et qu'il est prêt à l'écouter. L'interaction pourra alors commencer.

Dans le cas où le robot n'identifie pas la personne qui a dit "Hey Reachy", il fera le mouvement *triste*.

- Dans le cas où Reachy n'entend pas "Hey Reachy", le robot cherchera des interlocuteurs et les incitera à interagir avec lui, cela constitue le deuxième cas. En effet, avant qu'une personne ne rentre en interaction avec Reachy en lui disant "Hey Reachy", le robot suivra le visage de la personne passant le plus proche et fera, tout les 3 minutes et pendant 30s, le mouvement *triste*.

S'il détecte un utilisateur potentiel qui s'approche de lui, Reachy lèvera les antennes vers le haut et regardera la personne pour lui faire comprendre qu'il l'a bien vue et qu'il est prêt à l'écouter (mouvement *incitation*). Cela incitera la personne à communiquer avec le robot car il lui aura bien montré qu'il l'a détectée.

Après un "Hey Reachy" de la part d'un utilisateur le robot répondra "Bonjour" et l'interaction pourra commencer.

Dans un cas de nuisances sonores trop importantes, Reachy ne pourra pas identifier la commande vocale "Hey Reachy". Il sera alors possible pour un utilisateur extérieur de recourir à un marqueur ArUco¹ représentant un code afin d'indiquer à Reachy certaines actions. Ces actions seront en priorité l'initiation d'interaction, équivalent à prononcer "Hey Reachy" et à la prise de photo, simple ou en groupe.

Afin de permettre ces différents mouvements du robot nous allons utiliser l'API `reachy-sdk` de Pollen-Robotics développée pour le robot Reachy et par conséquence programmer en utilisant le langage *Python*. Elle permet de tourner la tête du robot vers la position demandée en une certaine durée. Ainsi que de bouger les antennes du robot à la vitesse souhaitée vers la position souhaitée. L'ensemble des positions, la durée et la vitesse sont paramétrables car ce sont des valeurs prises en paramètres des différentes fonctions.

3.4 Gestion des ordres

Pour donner des ordres à Reachy, les commandes devront contenir des mots clés spécifiques à chaque groupe de commandes.

Tout d'abord, la commande d'activation du robot nommée "Activation" dans le diagramme d'activité est paramétrable. Elle peut être définie par "Hey Reachy", "Ok Reachy", "Bonjour Reachy", etc . Une fois l'interaction débutée comme expliqué en section 3.3, le robot sera en attente d'un ordre (une commande lancée par l'utilisateur).

De manière générale, toutes les commandes, donc ensemble de mots clés et ensemble de couples entrée/sortie, seront spécifiées dans un ou plusieurs fichier(s) `.txt` afin de pouvoir les modifier sans avoir à toucher au code. Une commande à l'intérieur d'un fichier sera de la forme :

```
cmd : [nom de la commande]
. [mot clé]
. [mot clé]
...

```

Chaque commande aura donc un nom précédé de "cmd :" qui permettra à python de savoir de quelle commande il s'agit. Chaque mot clé sera précédé d'un ":" et le marqueur de fin d'une liste de mots clés est "...".

1. Les marqueurs ArUco sont identifiables avec la bibliothèque OpenCV et sont faciles à implémenter

Pour écouter les ordres, le robot fera le mouvement *écoute*. Si Reachy n'a pas compris la demande de la personne, il fera le mouvement *incitation* pour faire comprendre qu'il n'a pas compris et annoncera "Je n'ai pas bien compris votre demande". Aussi, si au bout d'un certain temps la personne ne pose pas de questions Reachy fera le mouvement *incitation* et il proposera d'énumérer ses fonctionnalités. En cas de non interaction trop longue, le robot fera l'émotion triste avant de se remettre à chercher un autre utilisateur dans son état initial.

Comme indiqué sur le diagramme d'états, le robot gérera des commandes pour la prise de photos qui seront détaillées dans la partie 3.5 et également des commandes de type conversation (cf 3.7).

A la suite d'une commande de type "Que sais-tu faire?", Reachy énumérera toutes ses fonctionnalités à l'utilisateur.

Pour réaliser la partie reconnaissance vocale, nous avons choisi d'utiliser la bibliothèque open source `SpeechRecognition` ainsi que `PyAudio` qui offrent des fonctions pour l'écoute, l'enregistrement, la reconnaissance vocale et la traduction du son en texte.

En ce qui concerne la réalisation de la partie synthèse vocale, nous avons choisi d'utiliser la bibliothèque `pyttsx3` qui est une bibliothèque python open source qui convertit le texte en parole. Elle fonctionne hors ligne et utilise le synthétiseur vocal `eSpeak`.

Afin d'être utilisé durant les 100 ans de l'ENSEIRB-MATMECA le robot devra être fonctionnel dans un environnement bruyant. Il devra donc effectuer une réduction du bruit extérieur. Pour cela nous avons choisi d'utiliser la bibliothèque `wave` afin de pouvoir gérer les fichiers d'extension ".wav" ainsi que `pydub` afin de réaliser le filtrage. Chaque enregistrement sera donc stocké dans un fichier `enregistrement.wav` qui s'écrasera à chaque fois et qui sera traité avec `wave`.

3.5 Prise de photos

OpenCV est une bibliothèque proposant de nombreuses fonctionnalités de vision par ordinateur, spécialisée dans le traitement d'images en temps réel. Cette bibliothèque sera utilisée pour toute utilisation des caméras du robot et permettra la reconnaissance de visages, fonctionnalité utilisée pour la prise de photo. Elle permettra également d'effectuer les différents filtres photos cités dans les fonctionnalités à implémenter après les 100 ans de l'ENSEIRB-MATMECA, en section 4.2.

Il y a deux types distincts de prise de photos, la photo simple d'une personne et la photo d'un groupe de personnes.

Pour entrer dans la partie prise de photo, la commande devra contenir l'un des mots {"photo", "selfie"}. Afin de spécifier le type de la photo, il y aura deux cas de figure : si la commande contient l'un des mots {"moi", "m'e"}, alors ce sera une **photo simple**. Et si elle contient un ou plusieurs mots clés suivants {"nous", "les", "groupe", "ensemble"}, alors ce sera une **photo de groupe**.

Exemple de commandes pour une photo simple : "Prends-moi en photo", "Hey Reachy, est ce que tu peux me prendre en selfie?" .

L'utilisateur pourra demander une photo sans préciser le type de prise de photo. Dans ce cas, Reachy invitera l'utilisateur à préciser s'il souhaite une photo simple ou de groupe. L'utilisateur pourra également demander une photo simple ou une photo de groupe directement.

Dans le cas où Reachy comprend qu'il doit faire une photo mais ne connaît pas encore quel type d'image est souhaité, il dira : "Vous préférez faire une photo simple ou une photo de groupe?" .

Après connaissance de la demande de prise de photos et du type de photo, Reachy ajustera la position de sa tête pour correctement cadrer la photo. Reachy cadrera la photo en conséquence du type demandé à l'aide des positions des visages obtenues grâce à la reconnaissance faciale d'OpenCV.

Dans le cas d'une photo simple, Reachy centrera son regard sur la position de l'unique visage à prendre en photo (le plus proche dans son champ de vision). Pour une photo de groupe, Reachy considérera tous les visages présents dans son champ de vision, il supprimera ceux estimés trop loin (trop petits par rapport à la moyenne des tailles des visages en vue) et il orientera son champ de vision vers la position moyenne des visages encore

concernés.

Ensuite Reachy prendra la photo par enregistrement d'un instantané sur la capture vidéo avec OpenCV, et ce juste après l'annonce d'un compte à rebours : "Je prends la photo dans 5..4..3..2..1.. CHEEESE!".

Des images représentant un marqueur ArUco pourront être utilisées afin d'indiquer à Reachy quel type de prise de photos l'utilisateur souhaite en cas de nuisance sonore trop importante.

3.6 Gestion des photos

Une fois que la photo a été prise, elle sera sauvegardée sur la mémoire du robot et ensuite affichée afin que les utilisateurs puissent la voir.

3.6.1 Sauvegarde de la photo

Une fois que les photos auront été prises, elles vont être enregistrées dans la mémoire du robot. Nous avons choisi de nommer la photo avec la date, l'heure et les minutes de quand elle a été prise, par exemple 08_04_2022-15_45.jpg. Cependant, sa mémoire étant limitée, il nous faudra donc gérer le fait de vouloir sauvegarder une photo supplémentaire lorsque qu'il n'y aura plus assez d'espace.

Dans ce cas, nous favoriserons les photos récentes à celles prises précédemment. Pour cela, lorsque l'enregistrement de la photo ne pourra pas être fait par manque d'espace mémoire, un message sera envoyé pour avertir les administrateurs. Les administrateurs exécuteront une commande permettant de supprimer le nombre souhaité des plus anciennes photos (suppression à l'aide du module python `os`). Cela nous permettra de libérer assez d'espace pour enregistrer de nouvelles photos.

3.6.2 Affichage de la photo sur un écran

En plus de pouvoir sauvegarder les photos, un besoin spécifique que les utilisateurs puissent voir la photo une fois qu'elle aura été prise pour voir si elle leur convient. Pour cela, la photo sera affichée en temps réel sur un écran connecté au robot via un câble HDMI.

3.7 Conversation

Dans un premier temps Reachy pourra entretenir une conversation très simple avec son interlocuteur. Cette conversation sera composée d'entrées et de sorties, par exemple le robot répondra "Bonjour" si on lui dit "Salut", "Bonjour" ou bien "Hello". Il répondra également qu'il va très bien si on lui pose la question mais on ne pourra pas converser de manière complexe car il ne retient pas l'état de la conversation (cela sera amélioré par la suite, cf 4.4). L'ensemble des couples entrées / sorties est précisé en annexe (cf 8.3).

Lorsqu'une personne sera en train de discuter avec le robot, il réagira à ce que la personne lui dit. Par exemple avoir l'air triste lorsque qu'une personne lui dira des mots considérés comme méchants et inversement content lorsqu'il entendra des mots considérés comme gentils. Si Reachy ne comprend pas les mots dit par la personne, il fera le mouvement *incitation* pour faire comprendre qu'il n'a pas compris.

3.8 Fin d'interaction

Lorsque l'interaction sera terminée, par la demande de l'utilisateur avec une commande ou si le robot ne reçoit plus d'ordre, Reachy remerciera la personne d'avoir interagi avec lui tout en faisant le mouvement *remerciement*.

4 Partie 2 : Après les 100 ans

Après les 100 ans de l'**ENSEIRB-MATMECA**, nous développerons les fonctionnalités liées aux filtres sur les photos, à la recherche par Internet ainsi qu'aux conversations. Ces fonctionnalités sont détaillées ci-dessous.

4.1 Diagramme d'états

Voici le diagramme d'état de la version finale du robot :



FIGURE 3 – Diagramme d'états final du Reachy Mini

4.2 Filtres sur les photos

Ensuite à la prise de photos, des filtres pourront être appliqués à ces photos :

- Le premier filtre consistera à faire de l'échange de visages, où les deux visages les plus proches sur la photo seront échangés.
- Le second filtre consistera à mettre la photo en noir et blanc.

Si le développement est assez avancé, les filtres pourront être appliqués en direct sur la capture vidéo, et une dernière fonctionnalité consistera à indiquer la similarité des deux visages les plus proches sur la photo. Ces fonctionnalités seront implémentées à l'aide de l'API d'OpenCV.

4.3 Recherche Internet

L'un des besoins fonctionnels du client est la recherche sur internet, notamment pour des conditions de surf à une date et un lieu précis. Ainsi nous utiliserons les API `webbrowser` et `request` afin de réaliser des recherches selon des urls et extraire des données trouvées sur une page web.

Si l'utilisateur réalise une demande contenant un des mots clés `{condition, surf}`, alors le robot lui demandera pour quel jour J et quel lieu L réaliser la recherche des conditions de surf. Reachy pourra ainsi effectuer une recherche internet en complétant l'url correctement avec J et L, et en extraire les informations importantes qu'il dira à l'utilisateur par l'intermédiaire de la synthèse vocale.

Ces recherches seront effectuées sur le site <https://www.surf-report.com/>, et le robot explicitera la hauteur des vagues, la température sur place ainsi que le temps.

Lorsque la personne aura demandé d'effectuer une recherche, il fera le mouvement *réflexion* ainsi que le mouvement *incitation* s'il n'a pas compris la demande de la personne.

4.4 Conversations plus interactives

Après les 100 ans de l'ENSEIRB la partie conversation du robot devra être améliorée. Le robot devra être capable de tenir une conversation simple avec son interlocuteur, il devra notamment pouvoir tenir une conversation du type :

- (I) Bonjour.
- (R) Hello !
- (I) Qui es-tu ?
- (R) Je suis Reachy et vous ?
- (I) Je m'appelle Test.
- (R) Enchanté Test
- (I) Comment vas-tu ?
- (R) Je me porte bien merci, et vous Test ?
- (I) Je vais très bien
- (R) Super
- (I) Que fais-tu ici ?
- (R) Je propose mes services de photographe pour l'événement d'aujourd'hui.

Pour cela nous allons continuer avec des mots clés mais en ajoutant une notion de mémoire de l'état de la conversation et comme l'indique le nouveau diagramme d'états on ne sort pas forcément directement de l'état Conversation à chaque commande. La figure 4 résume les interactions ajoutées :

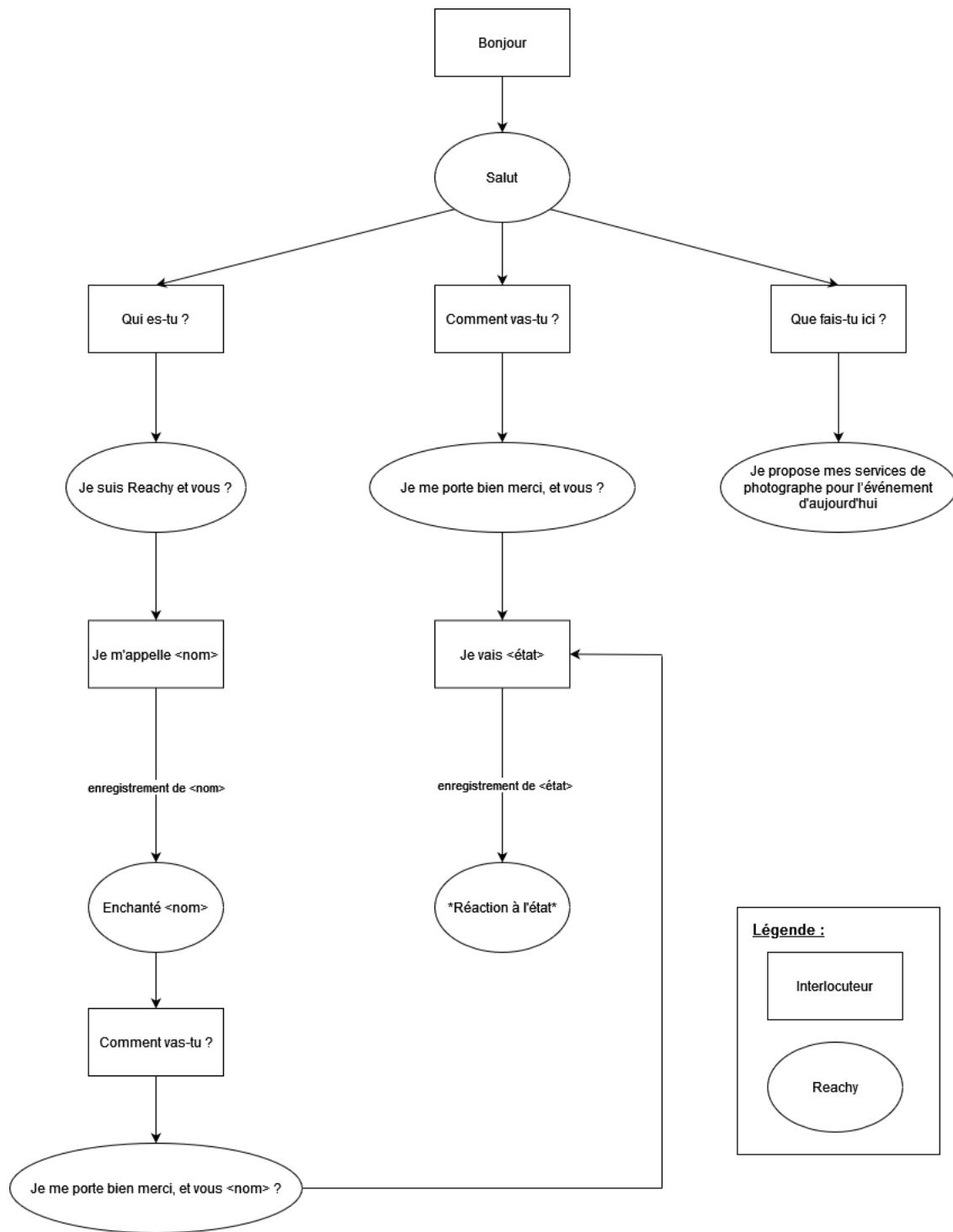


FIGURE 4 – Schéma des Conversations avancées

On peut voir les rectangles de la Figure 4 comme des noms d'ensembles de mots clés qui sont spécifiés en annexe (cf 8.2). Les ellipses quant à elles peuvent être vues comme les sorties car ce seront les phrases dites par le robot.

Cette partie conversation avancée pourra être densifiée de manière optionnelle, voici ce qui pourra être rajouté :

- Des couples entrée/sortie marrants comme par exemple : "Je suis ton père" / "Vous devez confondre, je ne suis pas Luke Skywalker".
- Des devinettes avec un fichier contenant un ensemble de couples question/réponses, lues grâce aux commandes natives de python. L'interlocuteur pourra demander une devinette, le robot choisira un couple

- question/réponses, posera la question et vérifiera la conformité de la réponses avec celle qu'il connaît (mots clés pour les réponses).
- Des questions de culture générale sur le même modèle que les devinettes.
 - Des blind tests sur des extraits audio de 10 secondes sur le même modèle que les devinettes.

5 Deux cas d'utilisation

Les principales fonctionnalités du robot peuvent être mises en application à l'aide de cas d'utilisation. La couverture du diagramme d'état est un indice de la force de ces cas d'utilisation. Il serait possible d'obtenir une pleine couverture en réalisant des cas d'utilisation pour chaque type de photo, et chaque type de conversation (recherche internet ou non, conversation simple ou non). Nous développerons ici les deux principaux cas qui permettent une couverture suffisamment élevée.

5.1 Interaction directe

Cas d'utilisation	Interaction directe
Description	L'utilisateur n'est pas détecté par le robot, il se dirige vers lui et lui demande de le prendre en photo.
Acteurs	- Un utilisateur - Reachy
Scénario	<ul style="list-style-type: none"> - L'utilisateur arrive et dit "Hey Reachy" au robot. - Reachy détecte la personne en redirigeant sa tête vers lui, fait le mouvement <i>écoute</i>. Il lui répond: "Hey, je vous écoute !" et se met en état d'attente d'ordre. - L'utilisateur lui répond: "Je veux faire une photo" - Reachy redresse sa tête et lui demande: "Vous voulez une photo simple ou une photo de groupe ?" - L'utilisateur précise qu'il veut une photo simple. - Reachy cadre la photo, vérifie que l'utilisateur est bien centré dans l'image, et ensuite se met à prendre la photo: "Je prends la photo dans 5 4 3 2 1 ... cheeeese !". - Reachy sauvegarde la photo, l'affiche sur l'écran connecté à lui et se met en état d'attente d'ordre. - L'utilisateur part sans dire au revoir. - Reachy dit: "Je vous écoute, si vous voulez connaître l'ensemble de mes capacités demandez moi en disant "Que sais-tu faire ?" - Reachy ne détecte plus la personne, il fait le mouvement <i>triste</i>, redresse la tête et se remet dans l'état de recherche d'interaction.
Pré-conditions	- Stockage non saturé

FIGURE 5 – 1^{er} cas d'utilisation

Dans un premier cas, on considère un utilisateur qui n'est pas détecté par le robot. Il adresse la parole à Reachy en lui disant "Hey Reachy". Reachy recherche donc l'utilisateur en tournant la tête vers lui.

Une fois la personne détectée, le robot, fait le mouvement *écoute*, et répond "Hey, je vous écoute!" . Il se met alors en attente d'un ordre de l'utilisateur. Ce dernier demande "Je veux faire une photo". Reachy redresse sa tête et demande : "Vous préférez faire une photo simple ou une photo de groupe ?". L'utilisateur précise qu'il veut faire une photo simple.

Le robot se met donc à cadrer la photo afin d'avoir l'utilisateur au milieu de l'image. Ensuite, il rentre en phase de prise de photo et lance le compte à rebours pour capturer la photo en disant : "Je prends la photo dans 5..4..3..2..1..CHEEESE!". Une fois la photo prise, elle sera sauvegardée et affichée sur l'appareil connecté au robot.

Reachy se retrouve alors de nouveau en attente d'un ordre. L'utilisateur est parti sans dire au revoir, donc le robot considère l'attente trop longue, et précise "Je vous écoute, si vous voulez connaître l'ensemble de mes capacités demandez moi en disant "Que sais-tu faire?"". Comme l'utilisateur n'est plus là, le robot fait le mouvement *triste* puis redresse la tête pour pour la recherche de visages qui est le premier état.

5.2 Interaction par détection de visage

Cas d'utilisation	Interaction après détection de visage
Description	L'utilisateur est détecté par le robot qui l'incite à interagir
Acteurs	- Un utilisateur - Reachy
Scénario	<ul style="list-style-type: none"> - Le robot recherche avec qui interagir, il fait des mouvements et des bruits mignons pour attirer l'attention. - Le robot détecte une personne, il bouge ses antennes, fait le mouvement <i>content</i> et dit "Bonjour !" - Le robot lui propose de dire "Hey Reachy" tout en faisant le mouvement <i>incitation</i> pour commencer l'interaction. - L'utilisateur dit "Hey Reachy" - Le robot lui répond: "Hey, je vous écoute" - L'interlocuteur lui demande: "Comment vas-tu ?" - Le robot lui répond: "Je vais très bien" - L'utilisateur lui demande: "Sais-tu préparer des cookies ?" - Reachy ne comprend pas la demande et répond: "Je n'ai pas bien compris votre demande" en faisant le mouvement <i>incitation</i>. - L'utilisateur lui répond méchamment: "Tu es nul et tu ne sers à rien !" - Reachy détecte des mots tristes et fait le mouvement <i>triste</i> (10 secondes). - Reachy redresse sa tête et attend l'ordre suivant. - L'utilisateur lui dit: "Au revoir !" - Reachy comprend que c'est la fin d'interaction et lui répond: "A bientôt" et réalise le mouvement <i>remerciement</i> - Reachy se remet dans l'état de base et recherche avec qui interagir.

FIGURE 6 – 2^{eme} cas d'utilisation

Dans ce deuxième cas, l'interaction est débutée par la détection d'un utilisateur par Reachy.

Le robot recherche une personne tout en faisant des mouvements et des bruits mignons pour attirer l'attention. Il détecte donc une personne à proximité, et se met à agiter les antennes avec le mouvement *content* et dit "Bonjour".

Il propose ensuite au client de dire "Hey Reachy" tout en faisant le mouvement *incitation* pour commencer l'interaction. L'utilisateur déclenche effectivement l'interaction en disant "Hey Reachy" et Reachy répond "Hey, je vous écoute". Ensuite l'interlocuteur demande "Comment vas-tu?", le robot lui répond alors "Je vais très bien".

Ensuite l'utilisateur demande : "Sais-tu préparer des cookies?". Reachy dans ce cas ne comprend pas et répond gentiment "Je n'ai pas bien compris votre demande" en faisant le mouvement *incitation*.

L'utilisateur enchaîne ensuite la conversation en disant "Tu es nul de toute façon!". Le robot identifie donc que ce sont des mots tristes, et fait le mouvement *triste* pendant 10 secondes.

Après il redresse sa tête et attend le prochain ordre ou la prochaine réplique. Le client lui dit "A bientôt", Reachy comprend donc que le client s'écarte. Il lui répond "A bientôt", réalise le mouvement *remerciement* et se remet dans l'état de base qui est la recherche de personne.

6 Tâches et livrables

Dans cette partie, nous détaillerons les étapes de la réalisation technique, ensuite, nous présenterons les diagrammes de Gantt et nous listerons tous les livrables.

6.1 Les étapes de réalisation techniques

6.1.1 La reconnaissance faciale

- **Identification de visage** [2 jours] : Obtention en capture vidéo d'une structure de donnée renseignant des informations sur la position des visages et affichage à l'écran clarifiant la réussite de l'identification (carré autour des têtes, points particuliers du visage, ...)
- **Suivi de visage** [1-2 semaine(s)] : Obtention des angles horizontaux et verticaux entre la normale à la caméra et le centre des visages de la capture vidéo. Adaptation des angles ressortis pour un suivi optimal du visage le plus proche.
- **Cadrage pour prise de photo** [1-2 semaine(s)] : Prise en compte des visages les plus proches, puis recherche du barycentre des centres des visages pour diriger le regard vers ce barycentre en guise de cadrage
- **Échange de visages** [1-3 semaines moyennant qualitativité] :
 - [3 jours] : Échange des deux têtes les plus proches (par les carrés les entourant)
 - [5 jours] : Échange des deux visages les plus proches (par une méthode plus poussée). Les visages n'auront pas les bonnes proportions
 - [1 semaine] : Correction des proportions et adaptation des traits faciaux
 - [5 jours] : Adaptation de la colorimétrie des visages échangés
- **Mise en noir et blanc** [1 jour] : Colorimétrie de la photo transformée en noir et blanc
- **Similarité de visages** [5 jours] : Pourcentage de similarité des deux visages les plus proches sur la photo

6.1.2 Mouvement du robot

- **Premier mouvement de tête et des antennes séparément** [1-3 jour(s)] : effectuer un mouvement quelconque avec la tête, de même pour les antennes.
- **Faire avoir des émotions** [1-2 semaine(s)] : Combiner des mouvements de la tête et des antennes pour faire comme si le robot avait des émotions ("content", "triste", "pensif", ...).
- **Faire avoir des émotions à partir d'une position initiale connue** [1-2 semaines] : Lui faire avoir les émotions dans l'orientation où la tête était avant ; pour cela lui donner les déplacements sous la forme de 3 angles qui seront ensuite transformés sous la forme de coordonnées cartésiennes.
- **Mouvement lié aux photos** [1 semaine] : Récupérer, de la partie photo, les valeurs des angles dont le robot doit bouger pour pouvoir prendre puis cadrer la photo.
- **Mouvement permettant de suivre les personnes** [1 semaine] : Tourner la tête pour qu'elle soit toujours en face de la personne qui, soit interagit avec Reachy, soit la personne la plus proche lorsque le robot n'interagit avec personne.
- **Mouvement pour se tourner vers le son** [1 semaine] : Récupérer, de la partie reconnaissance vocale d'où vient le son, les valeurs des angles dont le robot doit bouger pour pouvoir plus facilement trouver où se trouve la personne qui parle.
- **Fluidifier les mouvements** [2 semaines] : Faire que les mouvements soient plus fluides et plus naturels afin qu'ils se rapprochent plus de ceux d'un être humain.

6.1.3 Gestion des ordres

- **Transcription de la voix en texte** [1 jour] : écouter l'interlocuteur et transformer ses paroles en texte.
- **Atténuation du bruit extérieur moyen** [8-13 jour(s)] : atténuer les bruits extérieurs afin de se concentrer sur la source principale du son et pouvoir être utilisé dans un environnement moyennement bruyant
- **Mise en place d'un système de conversation basique** [3-5 jour(s)] : le robot doit pouvoir répondre à de commandes telles que "Bonjour" ou bien "Qui es-tu ?".
- **Gestion des marqueurs ArUco** [5-8 jours] : reconnaître un marqueur ArUco, le retranscrire en entier et l'associer à une commande existante pour pouvoir être utilisé sans reconnaissance vocale.

- **Atténuation du bruit extérieur fort** [9-14 jour(s)] : atténuer les bruits extérieurs afin de se concentrer sur la source principale du son et pouvoir être utilisé dans un environnement fortement bruyant.
- **Amélioration du système de conversation** [12-15 jour(s)] : le robot doit pouvoir tenir une conversation basique (cf 4.4).

6.1.4 Synthèse vocale

- **Conversion d'un texte en son** [1 jour] : faire saisir un texte et le transformer en parole.
- **Faire parler le robot** [4 jours] : vérifier les sorties audio du robot et ses paramètres et le faire parler.
- **Faire des petits bruits** [4 jours] : faire produire des petits bruits mignons et des phrases pour attirer l'attention.
- **Amélioration de la voix** [1 semaine] : améliorer la voix du robot.

6.1.5 Recherche Internet

- **Reconnaissance des informations nécessaires** [5 jour] : Transformer les paramètres énoncés par l'utilisateur en url de recherche.
- **Extraction des informations nécessaires** [1 semaine] : Extraire sur la page web les informations utiles (hauteur de vagues, température, temps).
- **Énoncer les informations** [1 semaine] : Énoncer ces informations à l'utilisateur de façon précise et concise.

6.2 Diagrammes de Gantt

Toutes ces étapes de la réalisation peuvent être résumées dans un diagramme, nommé diagramme de Gantt, qui permet de définir des dates précises concernant l'évolution du robot. De plus, cela permet la visualisation de livrables qui seront rendus pour réaliser des cycles de construction du projet et retour du client. Nous avons découpé ce diagramme en deux sous diagrammes, un concernant avant les 100 ans de l'**ENSEIRB-MATMECA** et un après.

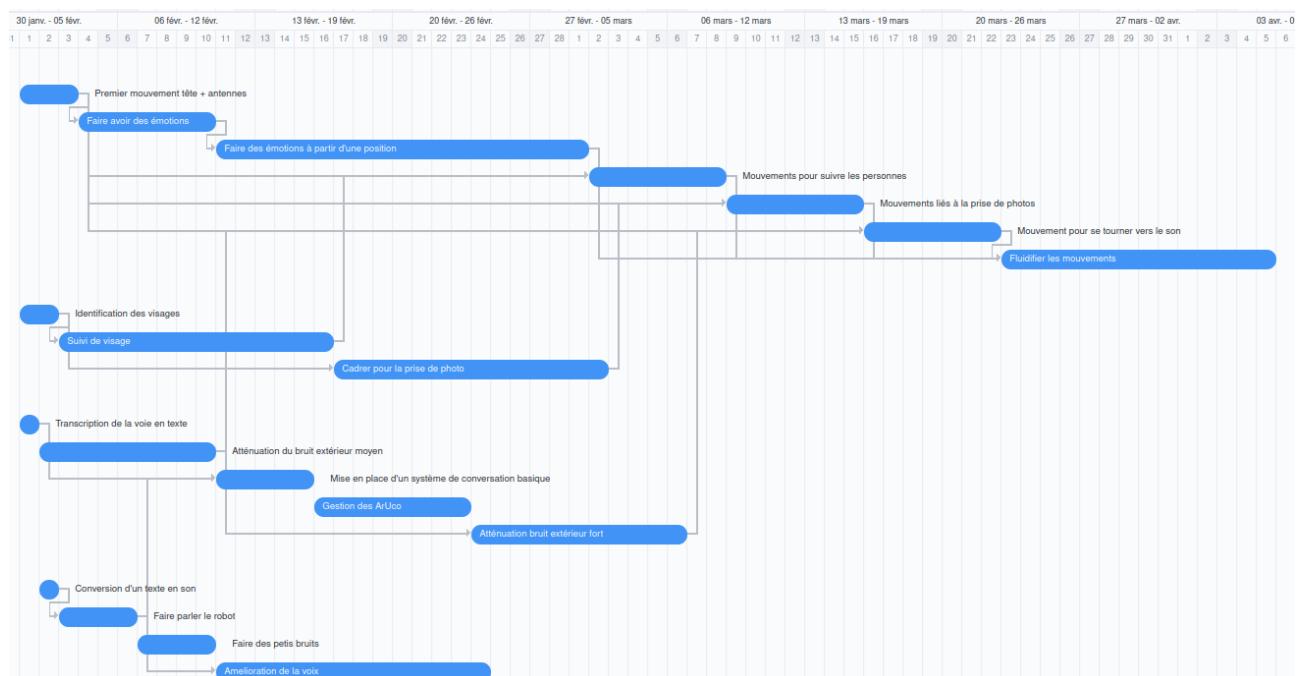


FIGURE 7 – Diagramme de Gantt avant les 100 ans

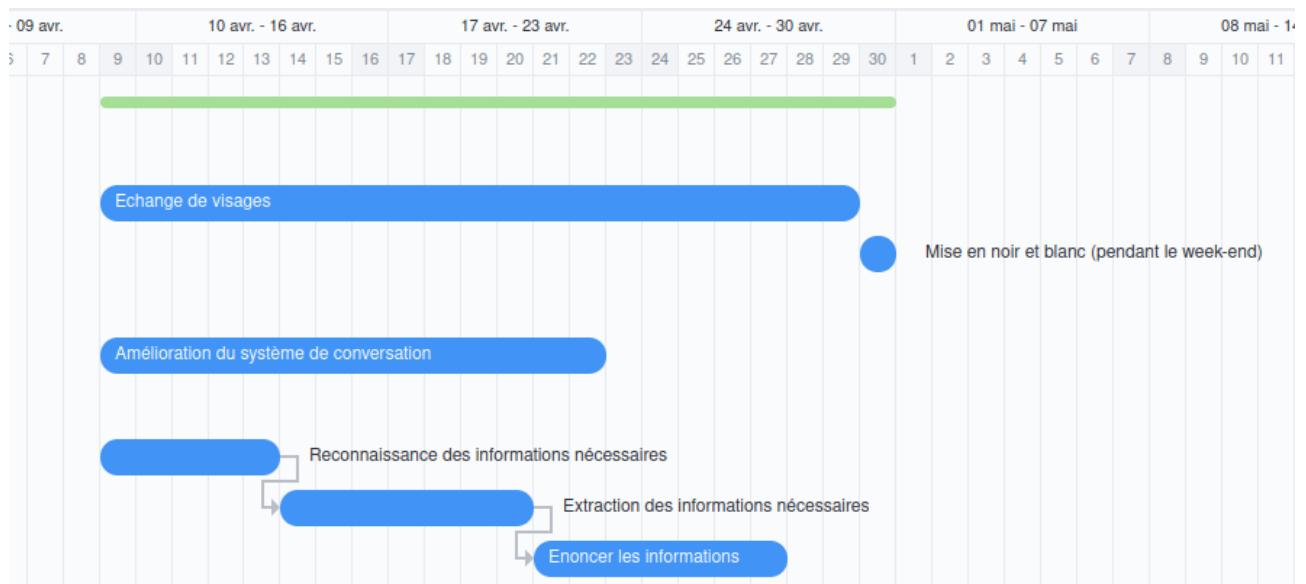


FIGURE 8 – Diagramme de Gantt après les 100 ans

6.3 Livrables

Voici les livrables qui se dégagent du diagramme de Gantt :

6.3.1 "Hello World" [07/02/22]

L'objectif de ce livrable est de fournir un robot capable d'effectuer les actions les plus basiques de chacun des axes majeurs de son fonctionnement. Il devra donc pouvoir effectuer un mouvement simple, reconnaître un visage, entendre une voix et la transformer en texte et dire une phrase quelconque.

6.3.2 Full Interaction [16/02/22]

L'objectif de ce livrable est de fournir un robot capable de lier ses axes majeurs de fonctionnement entre eux de manière basique. Il devra donc reconnaître une personne, bouger en conséquence, écouter une voix et répéter la phrase dite.

6.3.3 Le robot communique avec émotions [25/02/22]

L'objectif de ce livrable est de fournir un robot capable d'avoir une conversation basique avec l'utilisateur tout en montrant ses émotions.

6.3.4 Le robot sait prendre des photos [11/03/22]

L'objectif de ce livrable est de fournir un robot capable de cadrer une ou plusieurs personnes pour les prendre en photo.

6.3.5 Le robot communique et prend des photos [18/03/22]

L'objectif de ce livrable est de fournir un robot capable d'avoir une conversation plus dynamique avec l'utilisateur après l'avoir détecté et capable de le cadrer et le prendre en photo.

6.3.6 100 ans de l'ENSEIRB [01/04/22]

L'objectif de ce livrable est de fournir un robot capable d'effectuer tout ce qui est spécifié dans la partie 3. Les mouvements du robot sont plus fluides et le robot propose différents filtres et fonctionnalités lors de la prise d'une photo.

6.3.7 Système de conversation amélioré [23/04/22]

Le robot peut entretenir une conversation avec un utilisateur de manière fluide et interactive (mémorisation du prénom, action en fonction du sentiment)

6.3.8 Recherche internet pour des conditions de surf [30/04/22]

Le robot permet d'annoncer les conditions de surf pour une date et un lieu précis selon la demande de l'utilisateur.

6.3.9 Full version [07/05/22]

L'objectif de ce livrable est de fournir un robot capable d'effectuer tout ce qui est spécifié dans ce cahier des charges. Les filtres sur les photos auront donc été rajoutés.

7 Webographie

- [1] <https://www.pollen-robotics.com>
- [2] <https://github.com/pollen-robotics/reachy-sdk>
- [3] <https://docs.pollen-robotics.com>
- [4] <https://docs.python.org/fr/3/library/wave.html>
- [5] <https://opencv.org>
- [6] <https://pypi.org/project/SpeechRecognition>
- [7] <https://pypi.org/project/PyAudio>
- [8] <https://pypi.org/project/pydub>
- [9] <https://pypi.org/project/pyttsx3>
- [10] <https://towardsdatascience.com>
- [11] <https://docs.python.org/3/library/webbrowser.html>
- [12] <https://fr.python-requests.org/en/latest>

8 Annexe

8.1 Définition des positions

Une position est la combinaison d'un mouvement de la tête et des antennes. Le mouvement de la tête est réalisé en fonction de 3 coordonnées (x , y , z) qui déterminent un point de l'espace où Reachy regarde. Pour se rendre dans une certaine position, il est possible de déterminer la durée que prendra le mouvement. Le mouvement des antennes est caractérisé par une vitesse limite des antennes ainsi que deux angles qui correspondent au positionnement de chaque antenne.

Nous détaillons ici les différentes positions de notre robot, dont nous pourrons factoriser le code.

Position	Cordonnées	Durée	Angle Antenne Gauche	Angle Antenne Droite	Vitesse Antennes
écoute	(0.5, 0, -0.05)	2.0	0	0	/
triste	(0.5, 0, -0.3)	2.0	140.0	-140.0	70
content	(0.5, 0, -0.05)	2.0	± 20.0	± 20.0	300
incitation	(0.5, 0, 0.05)	1.0	35.0	-35.0	70
réflexion	(0.5, 0.15, 0.15)	1.0	-40.0	40.0	70
remerciement	(0.5, 0, -0.20)	0.5	-40.0	40.0	70

Le mouvement content est un mouvement où les antennes alternent leur position. Ainsi, elles alterneront en différé entre -20 et +20. Le mouvement remerciement correspond à la mise en position écoute, puis remerciement, puis écoute. Ainsi, le robot baisse la tête comme pour saluer son interlocuteur.

8.2 Ensembles de mots clé

Nous avons choisi de définir des ensembles de mots pour les commandes. La transition dans le diagramme d'états est faite selon le type de transition que l'on a choisi.

Voici les différentes transitions :

- "one in" = si la phrase de l'utilisateur contient un mot de l'ensemble de mots.
- "all in" = si la phrase de l'utilisateur contient tous les mots de l'ensemble de mots.
- "n in" = si la phrase de l'utilisateur contient n mots de l'ensemble de mots.

Ensemble "Hey Reachy" ("all in") :

- Hey
- Reachy

Ensemble "Photo" ("one in") :

- photo
- selfie
- photographie
- photographier
- cliché

Ensemble "unique" ("one in") :

- moi
- me

Ensemble "Groupe" ("one in") :

- nous
- groupe
- ensemble
- les

Dans le diagramme il y a une transition directe entre "traitement ordre" et "photo simple" ou "photo groupe". En fait il s'agit d'une double analyse de la phrase de l'utilisateur par Reachy.
Si la phrase est "prend moi en photo" on passera d'abord dans l'état "Photo sans précision" avec la détection de "photo" puis on passera dans l'état "Photo simple" avec la détection de "moi".

Exemples "Photo unique" :

- Prend **moi** en **photo**.
- Tu peux **me** prendre en **photo** ?
- Tu peux **me photographier** ?
- Prend **moi** en **selfie**.

Exemples "Photo groupe"

- Prend **nous** en **photo**
- Prend une **photographie** de **groupe**
- Fait un **cliché** de notre **groupe**
- Tu peux nous **prendre** en photo **ensemble** ?

Passons maintenant aux ensembles de mots clés concernant les fonctionnalités implémentées après les 100 ans de l'ENSEIRB.

Ensemble "N&B" ("one in") :

- noir et blanc
- monochrome

Ensemble "F-S" ("one in") :

- échangeant
- échange

Exemples "Photo N&B"

- Prend une **photo monochrome**
- Prend une **photo en noir et blanc**

Exemples "Photo F-S"

- Prend une **photo** avec un **échange** de visage
- Prend une **photo** en **échangeant** nos visages

Ensemble "internet" ("one in") :

- conditions
- surf

Ce cas est différent car il demande une date et un lieu on devra donc repérer une date qui pourra être spécifiée de cette façon, on repère plusieurs cas :

Cas pas de date :

On recherche les conditions de surf le jour actuel.

Exemple : "quelles sont les conditions de surf à Lacanau?"

Cas une date peu précise (jour de la semaine) :

On recherche les conditions de surf correspondantes au jour de la semaine précisé le plus proche de la date actuelle. Exemple : "quelles sont les conditions de surf Samedi à Lacanau?"

Cas une date précise :

On recherche les conditions de surf le jour spécifié.

Exemple : "quelles sont les conditions de surf le Jeudi 27 Janvier à Lacanau ?"

Cas pas de lieu :

On recherche les conditions de surf au Grand Crohot.

Exemple : "quelles sont les conditions de surf aujourd'hui ?"

Cas un lieu précis :

On recherche les conditions de surf lieu précisé.

Exemple : "quelles sont les conditions de surf aujourd'hui à Lacanau ?"

8.3 Couple pour la conversation basique

Nous n'avons pas pour objectif de permettre une longue conversation avec le Reachy, il s'agirait plutôt de couple d'entrées et de sorties sans garder en mémoire l'état de la conversation actuelle. Le Reachy utilisera la bibliothèque `random` afin de pouvoir gérer l'aléatoire, ici une sortie est présentée sous la forme du texte de sortie suivi du pourcentage de chance de renvoyer cette sortie entre parenthèse.

Entrée :

- Bonjour
- Coucou
- Salut
- Hello

Sortie :

- Bonjour (50%)
- Salut (50%)

Entrée :

- Comment vas-tu ?
- Comment ça va ?

Sortie :

- Très bien et vous ? (100%)

Entrée (mots clés ; "**one in**") :

- gentil
- sympathique
- mignon
- beau

Sortie :

- Merci (100%) + Mouvement : content

Entrée (mots clés ; "**one in**") :

- méchant
- moche
- inutile
- nul

Sortie :

- Mouvement : *triste*

Entrée incomprise

Sortie :

- Je n'ai pas bien compris votre demande ? (100%)

8.4 Ensembles de phrases/mots clés pour la conversation avancée

Ensemble "Qui es-tu?" ("one in") :

- Qui es-tu ?
- Qu'es-tu ?

Ensemble "Je m'appelle [nom]?" ("one in") :

- Je m'appelle [nom]
- Je suis [nom]
- [nom]

L'ensemble "Je m'appelle [nom]" fait suite à "Qui es-tu?" donc Reachy sait que son interlocuteur va lui transmettre son nom c'est pourquoi si la réponse est composée d'un seul mot, Reachy l'interprétera comme le nom de son interlocuteur.

Ensemble "Que fais-tu ici?" :

Cet ensemble est un peu particulier, il est composé de 2 ensembles de mots clés à détecter. En effet, la phrase doit contenir l'un des mots suivants :

- Fais
- Pourquoi

Ainsi que l'un de ces mots :

- ici
- là

Ensemble "Comment vas-tu?" ("one in") :

- Comment vas-tu ?
- Ça va ?
- Tout va bien ?

Ensemble "Je vais [état]" ("one in") :

- Je vais [état]
- [état]

L'ensemble "Je vais [état]" se rapproche de "Je m'appelle [nom]" puisque Reachy s'attend ici aussi au type de réponse puisqu'il fait suite à une question de Reachy, un état seul peut donc suffire.

La suite fait la liste des états possibles détectés par Reachy ainsi que les mots clés associés.

Ensemble "bien" ("one in") :

- bien
- très bien
- super
- ça va
- en forme

Réponse de Reachy (Réaction à l'état) : "Super" + mouvement *content*

Ensemble "mal" ("one in") :

- pas (afin de détecter les états du type "ça ne va pas")
- mal
- triste
- fatigué

Réponse de Reachy (Réaction à l'état) : mouvement *triste*

9.2 Fichier .json

```

1   {
2     "states": {
3       "allumage_robot": {
4         "action": "allumage_robot_func",
5         "transitions": [
6           {
7             "target": "recherche_interaction_only_aruco"
8           }
9         ]
10      },
11      "recherche_interaction_only_aruco": {
12        "action": "recherche_interaction_only_aruco_func",
13        "transitions": [
14          {
15            "target": "attente_ordre_only_aruco",
16            "predicat": "activation_aruco_det",
17            "action": "reset_activation"
18          }
19        ],
20        "timeout": {
21          "time": 120,
22          "target": "incitation_aruco"
23        }
24      },
25      "incitation_aruco": {
26        "action": "incitation_aruco_func",
27        "transitions": [
28          {
29            "target": "recherche_interaction_only_aruco"
30          }
31        ]
32      },
33      "eteindre": {
34        "action": "eteindre_func"
35      },
36      "attente_ordre_only_aruco": {
37        "action": "attente_ordre_only_aruco_func",
38        "transitions": [
39          {
40            "target": "traitement_ordre_only_aruco",
41            "predicat": "aruco_verif"
42          }
43        ],
44        "timeout": {
45          "time": 30,
46          "target": "recherche_interaction_only_aruco"
47        }
48      },
49    },
50    "module": "reachy_state_machine_100_ans_func"
51  }

```

Listing 1 – Machine à états factice afin de comprendre la structure des .json