Département Informatique

IUT de Clermont-Ferrand



3 Avenue Blaise Pascal,

63170 Aubière

Rapport de projet tuteuré

**Réalisation d'un logiciel permettant d'utiliser l'appareil LeapMotion pour lancer des scripts à partir de gestes prédéfinis reconnus par l'appareil.**



Réalisé par : Thomas BLANC, Yoann PERIQUOI

Emrick PESCE, Romain Olivier, Augustin LABORIE Réalisé du 09/11/2020 au 29/03/2021

Professeur tuteur : Laurent Provot Année 2020-2021

# AUTORISATION A DIFFUSER SUR L’INTRANET DE L’IUT

Le groupe composé de Romain OLIVIER, Thomas BLANC, Yoann PERIQUOI, Emrick PESCE et Augustin LABORIE autorise le DUT Informatique de Clermont-Ferrand à diffuser notre rapport sur l’intranet de L’IUT de Clermont-Ferrand.

# REMERCIEMENTS

Nous tenons à remercier Mr Laurent Provot, professeur au DUT Informatique de Clermont-Ferrand, pour avoir accepté d’encadrer le projet et pour l’aide qu’il nous a apporté durant la réalisation de celui-ci.

Nous tenons également à remercier Mr Bouhours Cédric, chef du département Informatique de l’IUT de Clermont-Ferrand, pour nous avoir conseillé lors de la première soutenance tenue le 18/01/2021.

# SOMMAIRE

[AUTORISATION A DIFFUSER SUR L’INTRANET DE L’IUT 1](#_Toc67085084)

[REMERCIEMENTS 2](#_Toc67085085)

[SOMMAIRE 2](#_Toc67085086)

[1. Introduction 4](#_Toc67085087)

[I. Présentation du projet tuteuré 5](#_Toc67085088)

[1. Contexte 5](#_Toc67085089)

[2. Objectifs du projet 5](#_Toc67085090)

[II. Organisation du Projet 6](#_Toc67085091)

[1. Mise place méthode Agile « light » 7](#_Toc67085092)

[2. Gestion de projet 7](#_Toc67085093)

[III. Analyse 7](#_Toc67085094)

[1. Diagrammes divers (classe, cas d’utilisation, séquence) 7](#_Toc67085095)

[2. Choix langage retenu 7](#_Toc67085096)

[IV. Réalisation de notre application 7](#_Toc67085097)

[1.Frontend 7](#_Toc67085098)

[2.Backend 7](#_Toc67085099)

[V. Difficultés rencontrées 7](#_Toc67085100)

[1.Difficultés techniques 7](#_Toc67085101)

[2.Nouveau langage 7](#_Toc67085102)

[3.Retards 7](#_Toc67085103)

[V. Bilan technique 8](#_Toc67085104)

[1. Réalisations 8](#_Toc67085105)

[2.Protocols expérimentaux 8](#_Toc67085106)

[VI. Conclusion 10](#_Toc67085107)

[1. Réalisation finale 10](#_Toc67085108)

[2. Evolution du projet dans le futur 10](#_Toc67085109)

[VII. Résumé en Anglais 10](#_Toc67085110)

[BIBLIOGRAPHIE et WEBOGRAPHIE 11](#_Toc67085111)

[LEXIQUE 11](#_Toc67085112)

[ANNEXES 11](#_Toc67085113)

## I. Introduction

Ce rapport s’inscrit dans le cadre du projet tutoré final obligatoire dans le cursus du DUT Informatique de Clermont-Ferrand.

Dans le cadre du projet nous nous devions de sélectionner parmi des projets proposés par les professeurs ou bien déterminer par nous même le sujet qui allais être traité. Le sujet a été déterminer par notre groupe et plus particulièrement par Romain OLIVIER et Augustin LABORIE. Par la suite Mr Laurent Provot à accepter d’encadrer notre projet.

Nous nous sommes intéressés à l’outils disponible au Club Informatique de notre département : le Leap Motion qui est un capteur du mouvement de la main et des doigts comme entrée, à l’instar d’une souris. Par la suite nous avons décidé de développer un logiciel permettant d’utiliser cet outil. Ce projet et cette application prendra le nom de HandyHand.

Nous allons donc nous demander comment utiliser la caméra Leap Motion afin d’exécuter des scripts.

Ce projet réside en la réalisation d'un logiciel permettant d'utiliser l'appareil Leap Motion pour lancer des scripts à partir de gestes prédéfinis qui seront alors reconnus par l'appareil. Nous devrons utiliser les informations fournit par le Leap Motion afin de les traduire et de pouvoir définir des gestes.

Pour réaliser cette application nous avons choisi de diviser notre équipe en deux. Une équipe sera responsable de la réalisation du Front-End et Back-End. Thomas Blanc et Augustin Laborie seront responsable de la réalisation des interfaces tandis que Yoann Périquoi et Emrick Pesce réaliseront la partie métier. Romain Olivier sera responsable de naviguer entre les deux équipes pour apporter son expertise dans les deux domaines.

## II. Présentation du projet tuteuré

### 1. Contexte

Le Leap Motion est un capteur permettant de virtualiser nos mains. Cela nous permet ainsi de lancer un traitement prédéfini lorsqu'un certain mouvement est reconnu.

Ce projet n'est pas une innovation puisque de nombreux autres logiciels sont déjà en circulation sur la plateforme de téléchargement d'application du Leap Motion (Leap Motion SDK). Cependant nous ne partons pas d'une base déjà existante pour réaliser notre propre logiciel, nous utilisons seulement la librairie fournie par les développeurs du Leap Motion.

Ce projet est réalisé durant la période de la pandémie mondiale du COVID-19. Les cours au DUT Informatique de Clermont-Ferrand sont assurés seulement en distanciel. Le groupe a donc dû s’organiser pour réaliser l’intégralité du projet en distanciel.

Dans la quête de se mettre le plus possible en situation réelle nous avons demandé à Mr Laurent Provot de se considérer comme l’initiateur de ce projet pour lequel il serait alors le client faisant l’appel d’offre.

### 2. Objectifs du projet

Nous devrons livrer un logiciel fonctionnel et déployable avec lequel sera disponible une interaction complète via une interface graphique et une interface en ligne de commande (ou CLI) avec le contrôleur LeapMotion. Ces interactions nous permettrons alors de pouvoir lancer un script (suite de commandes) et ainsi de réaliser des tâches à partir d'un seul mouvement.

Celui-ci devra permettre à l'utilisateur de :

* Gérer ses scripts (créer, modifier, lire, mettre à jour et supprimer les scripts et leurs gestes déclencheurs)
* Initialiser ses scripts grâce à des mouvements
* Gérer la connexion avec le LeapMotion
* Avoir un retour visuel de ce que perçoit le LeapMotion
* Lancer un script à tout moment grâce à un outil qui observe en permanence le flux vidéo et repère les gestes
* Enregistrer son environnement sur une base de données distante
* Récupérer son environnement depuis n'importe où grâce au serveur distant
* S'authentifier pour accéder à son profil
* Utiliser son propre profil en local avec une gestion "hors ligne"

Ces fonctionnalités sont décrites dans ce diagramme de cas d’utilisation :

Diagramme de cas d’utilisation de l’application HandyHand



*Figure 1 : Diagramme de cas d’utilisation de l’application HandyHand*

## II. Organisation du Projet

### 1. Mise place méthode Agile « light »

Lors de la mise en place du projet, Mr Laurent Provot nous a demandé de suivre une gestion de projet particulière. En effet, celui-ci nous a demandé de suivre une méthode très populaire dans le domaine du développement de logiciel appelé la méthode Agile. Seulement celle-ci serait allégé, d’où l’adjectif anglais « light », c’est-à-dire que tous les rouages ne seront, pas ou peu abordé. Cette méthode réside en la mise en place de « sprints ». Chaque script commence par la définition d’histoire ou « backlogs » qui nous permettent d’énumérer les différentes tâches qui vont devoir être réalisé tout au long du projet.

Dès le début du projet toutes les tâches nécessaires à la réalisation de celui-ci été sont dès à présent défini. Par la suite, lors du démarrage de chaque sprint « sprint » lors de la réunion de démarrage, les « backlogs » ou tâches vont alors être sélectionné pour être réaliser lors de la période. Nos périodes de « sprint » été de deux semaines, avec une réunion médiane au début de la deuxième semaine. A la fin de ces deux semaines avais alors lieu une « démonstration » où nous avions la mission de présenter ce qui avais été réalisés pendant les deux dernières semaines. Nous faisions alors le bilan et Mr Provot nous donner alors des retours sur les travaux qui avaient été réalisé comme le ferai un client.

Nous avions étudié cette méthode de travail grâce au livre « Scrum depuis les tranchées » écrit par Henrik Kniberg mis à disposition sur le site web de Mr Provot. Il nous a permis d’apprendre les bases de cette méthode et de la rendre efficace même si c’était la première fois que tous les membres du groupe l’utilisaient.

Cette organisation nous a permis de suivre nos objectifs tout au long du projet.

### 2. Gestion de projet

### 

Lors de cette période une matière consistant en la gestion de projet nous a aussi aidé à apporter de l’organisation au projet. En effet, il nous a été demandé de réaliser de nombreux diagrammes de Gantt ainsi qu’un cahier des charges. Un extrait de celui-ci est d’ailleurs utilisé dans la définition des objectifs du projet. Nous allons donc voir

## III. Analyse

### 1. Diagrammes divers (classe, cas d’utilisation, séquence)

### 2. Choix langage retenu

## IV. Réalisation de notre application

### 1.Frontend

### 2.Backend

#### a. Reconnaissance

Notre projet se base sur le LeapMotion qui nous renvoie certaines informations sur les mains qu’il repère. Cependant, il ne nous permet pas de faire une reconnaissance de gestes que l’utilisateur effectue. En effet, les principales informations qu’il renvoie sont sur les positions et orientations des mains dans l’espace. Ainsi, pour mener à bien notre projet, il a fallu que l’on crée nous même une reconnaissance des gestes que l’utilisateur peut effectuer.

La première tentative fut de calculer différentes informations sur la main et ses doigts, à partir des différentes positions que l’on recevait du LeapMotion.

Nous allons en premier lieu vous présenter les informations utiles sur le LeapMotion.

Le LeapMotion nous permet de récupérer différentes coordonnées des mains, doigts et même plus. Pour que ceci soit possible, il utilise un système de coordonnées cartésiennes, avec les distances en millimètres. Ce système est le plus connu du grand public en mathématiques. Pour expliquer simplement, c’est un repère (par exemple , avec les axes de celui-ci), où les coordonnées d’un point seront exprimées suivant les axes de celui-ci. Par exemple, pour un point M, on aura des coordonnées , où et représente une certaine position sur les axes du repère. Le repère est en trois dimension, et son origine, c’est-à-dire son point de coordonnées , se trouve centré sur le dessus du LeapMotion, comme le montre cette illustration :



Une main est constituée de doigts, qui sont eux-mêmes constitués d’os. En soit, une main est composée de 27 os, cependant, les carpes, c’est-à-dire ceux situés au début de la main, proche du poignet ne sont pas pris en compte par le LeapMotion. En effet, il nous renvoi des informations sur quatre types d’os présents dans les quatre doigts :



Pour être plus précis, le pouce possède uniquement trois os, cependant, pour une histoire de simplicité du code, il est considéré avec quatre os, dont un, le métacarpe, avec une longueur de 0. Ainsi, pour résumer, le LeapMotion considère qu’une main possède cinq doigts, et que chacun de ceux-là possède quatre os, la phalange distale, la phalange intermédiaire, la phalange proximale et le métacarpe. Par conséquent, il nous renvoie, pour chacun des os, la position en son centre ainsi que celle de ses extrémités.

Au-delà de ces os, il renvoie également la position du centre de la main, qui est définie comme étant le milieu de la paume de la main. De plus, à partir de ce même point, il va nous donner la direction et l’orientation de la main. Pour ceci, il utilise un système de vecteurs, avec celui pour la direction (la flèche allant vers la droite sur le schéma), ainsi que le vecteur normal de la paume définissant ainsi l’orientation (la flèche allant vers le bas).



On remarque ainsi que le LeapMotion renvoi des informations utiles pour différentes utilisations.

Le LeapMotion est assez complet, cependant, il ne donne pas non plus toutes les informations dont nous avons besoin. En effet, dans notre quête de la reconnaissance de certains gestes, nous devons ajouter des apports personnels. Ainsi, au départ, pour permettre de reconnaître un geste codé, nous avons fait en sorte de créer des informations complémentaires sur la main.

Premièrement, nous avons voulu repérer des gestes simples, tels que la pierre, la feuille, et les ciseaux. Ces trois formes peuvent être mises en commun à partir de la fermeture des doigts. En effet, pour la pierre, il faut que tous les doigts soient pliés, pour la feuille il n’en faut aucun, et pour les ciseaux il faut uniquement l’index et le majeur. Ainsi, il a fallu trouver un moyen pour arriver à savoir lorsqu’un doigt est plié ou tendu. Pour cela, on s’est intéressé à des distances entre certaines positions de certains os. Lorsque les doigts sont courbés, ils se rapprochent du centre de la main. Par conséquent, le bout de l’os distal (qui se trouve au bout des doigts) va venir proche du centre de l’os métacarpien (celui au niveau de la paume) du même doigt (le pouce étant une exception que l’on verra après). Ainsi, on peut calculer la distance entre ces deux positions. Pour ce faire, cela est assez simple, on fait la distance entre deux vecteurs, ici dans R3 (c’est-à-dire dans un plan de dimension 3), qui nous est donné par la formule :

Avec le premier vecteur et le second vecteur.

Le LeapMotion étant dans un repère basé sur les millimètres, les distances sont alors en millimètres également. A partir de ceci, il a fallu faire plusieurs tests pour comparer les distances calculées par rapport à la courbure des doigts. Ces tests consistent à noter la distance séparant les deux vecteurs des os que l’on venait de calculer. Ceux-ci étant fait pour tous les doigts séparément, hormis le pouce, qui est différent, on a pu récupérer les distances lorsque les doigts étaient ouverts et fermés. A partir de ces données, nous avons décidé de trouver le pourcentage de courbure des doigts. Pour ce faire, voulant un pourcentage partant de zéro, on a fait partir nos données de zéro, c’est-à-dire que nous avons, pour chaque doigt séparé, soustrait la valeur de la distance lorsque le doigt est plié aux deux valeurs que nous avions. Ainsi, les données du doigt tendus ont été soustrait des données lorsque le doigt est plié, et ces données du doigt courbé sont passées à 0. Maintenant, il ne nous reste plus qu’à faire un produit en croix pour pouvoir trouver le pourcentage.

Le produit en croix (ou règle de trois) permet de trouver une quatrième valeur proportionnelle aux trois premières. Ceci est très connu en mathématiques, et est souvent utilisé dans la vie de tous les jours, que ce soit consciemment ou non. Prenons l’exemple du calcul d’un prix à payer en fonction du poids. Nous avons des champignons coûtant 5€ le kilogramme. Cependant, nous ne voulons pas acheter un directement, mais plutôt 200 grammes, soit 0.2 . Modélisons cela sous la forme d’un tableau :

|  |  |  |
| --- | --- | --- |
| Quantité () | 1 | 0.2 |
| Prix (€) | 5 |  |

Ici, est ce que l’on va devoir payer pour nos 200g de champignons.

Pour trouver cette valeur, nous multiplions la valeur à côté de l’inconnue (étant dans notre cas) par celle au-dessus, puis nous divisons cela par la troisième valeur connue. Ce qui nous donne :

Donc ici, , nous avons trouvé le prix que l’on devra payer.

Pour être plus précis, dans notre tableau, nous avons qui est proportionnel à .

|  |  |
| --- | --- |
|  |  |
|  |  |

Ceci implique alors l’égalité suivante :

Par conséquent, si nous cherchons la valeur de d, il nous suffit de diviser les deux parties par a, pour retirer a à gauche, ce qui nous donne :

Nous avons alors calculé d. Ce qui nous ramène donc à nos champignons, ce qui est bien ce que nous avons fait pour trouver l’inconnue.

Dans notre cas, nous avons la distance entre deux points. Pour faciliter l’explication, on va nommer la valeur maximale de la distance que nous avons trouvée, celle pour quand le doigt est tendu, par , la distance courante que renvoie le LeapMotion par , et le pourcentage de courbure du doigt par . On peut modéliser cela par un tableau :

|  |  |  |
| --- | --- | --- |
| Distance (mm) |  |  |
| Courbure (%) |  |  |

Le pourcentage de courbure de est à 100%, dans la mesure où pour , on estime le doigt tendu. Pour récapituler, nous connaissons les valeurs de et , et notre inconnue est . En appliquant le produit en croix vu précédemment, nous obtenons l’équation :

Ainsi, nous trouvons , le pourcentage de courbure du doigt, suivant , la distance entre le bout de l’os distal et le centre de l’os métacarpe.

Après ces calculs, nous devons mettre les valeurs trouvées à l’échelle. Ceci est nécessaire à partir du moment où on compare des valeurs brutes que l’on obtient après des tests, ce qui fait que cela n’est pas forcément généralisable si on a des mains plus grandes par exemple. Cela est aussi utile dans la mesure où le LeapMotion peut parfois renvoyer des valeurs incorrectes. Par conséquent, après ces calculs, nous devons les mettre à l’échelle. Pour ce faire, nous mettons à les pourcentages strictement inférieurs à zéro, et à les pourcentages strictement supérieurs à cent.

Récapitulons la démarche suivie :

* Nous avons fait des tests sur chaque doigt pour trouver une valeur approximative du maximum et minimum de la distance entre le bout de l’os distal et le centre de l’os métacarpe. On peut noter le maximum par et le minimum par .
* Nous avons fait en sorte que le minimum soit à 0, tout en gardant le même écart avec le maximum. D’où les calculs suivants :

Ceci nous permet que, lorsque l’on fait le produit en croix, le pourcentage soit bien à 0 pour la distance minimale (la valeur de base)

* Ensuite, nous avons fait un produit en croix permettant de calculer le pourcentage avec cette équation :

Avec le pourcentage recherché, la distance courante que vient de nous renvoyer le LeapMotion et la valeur maximale de la distance trouvée après les tests et après avoir été modifiée.

* Enfin, on met à l’échelle le pourcentage que l’on a obtenu, pour bien que les valeurs soient comprises entre 0 et 100.

Notre objectif de base est de savoir si un doigt est plié ou tendu. Maintenant que nous avons son pourcentage de courbure, cela nous rend la chose plus aisée. En effet, on sait que quand le pourcentage est à 100%, le doigt est plié, et quand il est à 0%, le doigt est tendu. Cependant, ces valeurs peuvent varier suivant la reconnaissance du LeapMotion, ou encore suivant la forme des mains de la personne. Par conséquent, on va plutôt définir que le doigt est plié lorsque le pourcentage est supérieur à 85%, et qu’il est tendu lorsqu’il est inférieur à 15%. Ainsi, nous savons que le doigt est plié, tendu ou encore aucun des deux.

Toutes ces informations collectées ne sont pas valables pour le pouce. En effet, ce doigt est bien différent des précédents. Rappelons comment est fait le pouce en utilisant ce schéma :



On peut définir que le pouce bouge de deux manières différentes. La première est lorsque l’os distal se penche en avant pour se rapprocher de l’os proximal de manière parallèle. La seconde est quand le pouce entier se dirige vers le centre de la main, c’est-à-dire que l’os proximal bouge. Ainsi, on va calculer le pourcentage de courbure de ces deux mouvements séparément. On pourra faire la moyenne de ces résultats si nous le souhaitons, ou encore les utiliser indépendamment. Le principe utilisé précédemment reste le même, seules les positions utilisées ne sont pas les mêmes. En effet, pour le premier mouvement, on prend le bout de l’os distal, ainsi que le centre du proximal. Pour le second, nous prenons le bout du proximal ainsi que le centre de la main, soit le centre du métacarpe du majeur.

Nous pouvons maintenant utiliser ces nouvelles données pour reconnaître des gestes simples. En effet, comme dis précédemment, si nous voulons savoir si la main est en forme de Pierre, il nous suffit de savoir si tous les doigts sont pliés. De même pour la feuille, il faut les doigts tendus, et pour les ciseaux, on doit avoir l’index et le majeur tendu, et tous les autres doigts pliés.

Notre objectif de reconnaissance de gestes simples étant atteint, nous souhaitons aller plus loin avec des gestes un peu plus complexes, tels qu’avec la forme d’un cœur avec deux mains ou encore le signe ok à une main, où l’index et le pouce se touchent.

En analysant ces deux gestes, on remarque facilement qu’il va nous falloir une méthode pour savoir quand plusieurs doigts se touchent. Cependant, avant, il nous faut quelque chose d’un peu moins visible, qui est la dissociation de la courbure des doigts en deux. En effet, on peut remarquer que pour former le cœur, le bout du doigt doit être plié, tandis que le début doit être tendu. Pour expliquer cela avec les os, nous devons avoir la phalange distale devant et parallèle à la phalange proximale, tandis qu’en même temps l’os métacarpe doit être dans la prolongation de la phalange proximale, comme pour former une droite.

Pour ce faire, nous reprenons la même méthode que précédemment, en calculant la courbure du bout des doigts grâce à la distance de la fin de l’os distal avec le centre de la phalange proximale. De même pour le début des doigts, avec le bout de l’os proximale et le début du métacarpe. On définit alors si ces parties sont pliées ou tendus suivant les pourcentages 85 et 15 comme précédemment, et pour ne pas perdre les premiers gestes créés, nous définissons la courbure du doigt entier en tant que la moyenne de ces deux parties.

Nous devons maintenant savoir lorsque des doigts se touchent. On peut remarquer sur le geste du cœur qu’il ne nous faut pas seulement savoir lorsque les bouts de deux doigts se touchent, mais aussi lorsque le début de la phalange distal se touche. Ainsi, il va falloir faire ceci pour les différents endroits où nous en aurons besoin, mais la démarche reste la même. Pour ce faire, on reprend la même démarche pour calculer une distance entre les positions que nous souhaitons, et nous affichons cette distance. Le fait de l’afficher, nous pouvons voir la distance approximative lorsque les doigts sont collés. Il ne nous reste plus qu’à définir que quand la distance est inférieure à celle trouvée, alors les positions sont collées. On peut faire de même pour savoir si le pouce est proche ou non de la main, pour d’autres gestes.

Si nous réfléchissons sur ce que nous avons fait jusque-là, ou encore si nous faisons différents tests sur les reconnaissances, nous pouvons remarquer qu’elle n’est pas suffisante. En effet, le pourcentage de courbure des doigts n’est pas toujours correct, et peu précis, ce qui bloque l’avancée de la reconnaissance. De plus, le fait d’utiliser des valeurs brutes trouvées à la main n’est pas forcément bon. Ceci nous amène au fait que nous devons améliorer la reconnaissance. Pour ce faire, nous avons eu l’idée de calculer la courbure des doigts d’une différente manière. En effet, nous allons calculer les angles entre les différents os pour la calculer.

Pour se faire, rappelons que le LeapMotion renvoie plusieurs positions sur chaque os, c’est-à-dire le début, le milieu et la fin. De plus, pour deux os qui se suivent, la position de fin de l’un est celle de début de l’autre. Par exemple, la position du bout de la phalange intermédiaire, est la position de départ de la distale. Nous pouvons ainsi voir chaque os comme un vecteur.

La notion de vecteur est le fondement de la branche des mathématiques appelée algèbre linéaire. Un vecteur à n éléments est la donnée ordonnée de n nombres réels (séparés par des points-virgules). Dit autrement, un vecteur permet de réunir plusieurs nombres réels (c’est-à-dire un entier ou un nombre à virgule), qui sont généralement une partie d’une droite orientée vers une position. Par exemple, dans notre cas, nous avons des vecteurs avec 3 nombres car notre repère contient 3 axes (ceci n’est pas forcément tout le temps cas, avec les vecteurs homogènes par exemple, mais nous n’allons pas nous attarder dessus pour le moment). Un vecteur peut se représenter de deux manières :

Le LeapMotion nous renvoie des vecteurs positions pour les os, c’est-à-dire que l’on possède uniquement certaines positions de l’os. Cependant, à partir de celles-ci nous pouvons calculer les vecteurs directeurs des os, soit représenter l’os comme une droite directive. Pour expliquer la démarche suivie, nous pouvons le faire avec la phalange distale ainsi que l’intermédiaire. Évidemment, ceci est également à faire sur les autres os qui sont collés entre eux. Voici un schéma pour mieux comprendre :



Nous nommons la position entre les deux os comme étant le point , le point est la position de la fin de la phalange distale, et le point le début de l’intermédiaire. Nous allons créer deux vecteurs à partir de ces points, le vecteur , et le vecteur . Pour créer un vecteur à partir de deux points, nous avons la formule suivante :

*Avec et les positions des points sur l’axe , de même pour et sur l’axe , et et sur l’axe .*

Par la suite, nous avons une équation qui nous permet de trouver le cosinus entre deux vecteurs. La fonction cosinus est une fonction qui donne à partir d’un angle en radian un certain nombre exprimé à partir de la valeur Pi (π).



La valeur retournée par la fonction cosinus, est la position du point noir sur l’axe , soit l’axe horizontal.

Cette suite d’images est plus parlante. On peut voir que suivant l’angle représenté, la position sur la courbe du cosinus change, et donc également la valeur retournée. L’intersection de la droite verticale avec l’axe horizontale à l’intérieur du cercle correspond au point noir du schéma de gauche. Si la droite se trouve à gauche, la valeur est négative, tandis que si elle se trouve à droite, elle sera positive.

Ceci nous amène au fait que nous possédons une équation permettant de trouver le cosinus de l’angle entre deux vecteurs.

Décomposons l’équation pour que ce soit plus simple. La partie :

Est le produit scalaire des vecteurs et . Pour mettre des mots simples dessus, on peut dire que c’est la somme des éléments de chacun des vecteurs multipliés entre eux. Ceci peut être ramené, dans notre cas, à l’équation :

Ensuite,

Est le module (ou la norme) du vecteur , c’est-à-dire sa longueur. Le produit scalaire du vecteur par lui-même donne le carré de son module, soit :

Ainsi, on peut trouver :

Enfin,

Représente la multiplication des modules des vecteurs. Par conséquent, avec toutes ces valeurs, nous avons trouvé . Cependant, nous voulons l’angle entre et , c’est-à-dire ce qu’y est passé dans la fonction cosinus. Par conséquent nous voulons la réciproque de la fonction cosinus. La réciproque est une application (un calcul par exemple) qui associe à chaque élément de l’ensemble d’arrivée d’une fonction à son unique antécédent par la fonction. En d’autres mots, pour une fonction , la réciproque permet de trouver à partir de la valeur . Si la réciproque est une fonction, que l’on peut nommée , alors on peut écrire :

Où est une fonction quelconque, et est sa fonction réciproque.

La fonction cosinus possède une fonction réciproque, nommée Arc cosinus, et écrite . Nous n’allons pas nous intéresser à son calcul dans la mesure où celui-ci est délicat à aborder, impliquant le logarithme complexe, mais aussi par forcément utile dans notre contexte. Cependant, cette fonction nous permet de trouver l’angle radian (le radian étant une unité de mesure d’angle) de nos deux vecteurs. Ceci revient à faire

Avec étant l’angle en radian formé par les vecteurs et .

En reprenant la formule précédente :

Avec étant l’angle en radian formé par les vecteurs et .

Par conséquent, nous venons de trouver l’angle radian étant entre les vecteurs et .

Maintenant que nous possédons l’angle radian, nous voulons chercher l’angle en degré (une autre unité de mesure d’angle) afin que ce soit plus simple d’utilisation. Pour cela, nous savons que :

Par conséquent, il suffit de multiplier la valeur en radian que nous avons trouvé par pour trouver la valeur en degrés de l’angle.

Nous avons alors la mesure de l’angle en degré pour chaque couple d’os. Cependant, le LeapMotion ne renvoyant pas des valeurs totalement exactes, l’angle n’est pas forcément correct, dans le sens où il y a un décalage moyen de plusieurs dizaines de degrés pour certains couples d’os. Par conséquent, bien que cela ne soit pas forcément la meilleure des choses, nous sommes contraints à refaire des tests comme au départ, pour trouver entre quelles valeurs d’angles chaque couple d’os oscille, pour ensuite calculer un pourcentage à partir de cela.

Nous pouvons alors maintenant faire la moyenne des degrés des couples d’os distal et intermédiaire, avec intermédiaire et proximal pour trouver le pourcentage de courbure des bouts des doigts. Nous avons également cette valeur pour le début des doigts avec le couple proximal et métacarpe. Ainsi, nous sommes revenus avec les mêmes valeurs que précédemment, mais celles-ci plus précises et fonctionnelles.

Bien qu’elle nous ait été utile pour faire différents tests, la reconnaissance précédente n’est pas optimale. En effet, celle-ci est contraignante pour différentes raisons.

Tout d’abord, pour que le LeapMotion reconnaisse un geste dit complexe (avec par exemple deux mains retournées côte à côte, et dont certains doigts sont pliés et d’autres non), il va falloir s’y reprendre à plusieurs fois, voir même faire une certaine séquence de mouvements pour bien que le LeapMotion remarque le geste correctement (par exemple montrer ses deux mains côté face, plier ses doigts, tourner ses mains et enfin les mettre côte à côte). Ceci est alors complexe et non intuitif pour tout utilisateur.

Ensuite, si nous voulons ajouter des méthodes de reconnaissances, telles que la détection de l’espace entre deux doigts, il faudrait réfléchir à comment faire, pour ensuite y coder, et enfin partager le code dans une mise à jour. Cependant ceci, bien que réalisable, n’est pas la meilleure, ni la plus simple pour l’utilisateur, des solutions.

Enfin, nous avons le qu’il faut coder chaque geste « à la main ». Ceci dans le sens où il faut dire comme tel ou tel doigt doit être plié, ou encore si les mains doivent êtres collés, ainsi de suite. Alors effectivement nous pourrions proposer une interface utilisateur, où chacun pourrait créer son propre geste, mais ceci ne serait pas intuitif à tout le monde, et d’ailleurs très limitant pour chacun.

Tout ceci nous amène à devoir réfléchir à une meilleure manière de reconnaissance. Celle-ci devra pouvoir laisser l’utilisateur créer son geste personnalisé, qui sera également conforme à la taille de sa main et facile à utiliser. Étant donné que le LeapMotion nous renvoie des coordonnées de différentes parties de la main, on pourrait se dire qu’il est possible de toutes les comparer avec un autre geste.

Les coordonnées sont des positions dans l’espace, ce qui veut dire qu’il faudrait que le geste soit exactement au même endroit, ce qui est très contraignant. Il faudrait alors trouver un moyen qui puisse faire en sorte que la position ne rentre pas en compte.

Ensuite, ceci prendrait en compte uniquement une certaine taille de main. Évidemment, on peut dire que l’on ne veut pas exactement les mêmes coordonnées, donc différentes tailles peuvent être reconnues, cependant ceci n’est pas suffisant. En effet, pour prendre par exemple la taille de mains d’enfants, il faudrait mettre un écart entre le geste voulu et le geste effectué suffisamment grand pour que le geste soit reconnu, mais ceci empiéterait sur la précision du geste (par exemple on ne serait pas à même de différencier un cœur d’un rond fait avec les deux mains).

Enfin, la rotation de la main est prise en compte dans les coordonnées, tandis que nous voulons que le geste soit reconnu quel que soit la direction de la paume de la main (à condition que le LeapMotion puisse détecter correctement la main complète).

Ces trois points peuvent se résoudre respectueusement avec la translation, la mise à l’échelle, et la rotation.

La translation permet, pour un objet géométrique, de déplacer tous ses points de la même distance, avec la même direction et le même sens. Dans notre cas, ceci signifie changer la position initiale de la main vers un autre endroit. Le principe de la translation est assez simple. Imaginons un cube avec 8 sommets (les coins), dans un repère de dimension 3.



Le cube



Le repère

Si nous souhaitons faire descendre le cube, alors chaque sommet de celui-ci devra bouger vers le bas sur l’axe du repère, c’est-à-dire qu’ils devront chacun diminuer leur position . Par exemple, si nous voulons faire descendre le sommet de dix unités, alors nous ferons :

Avec les coordonnées de base de .

Ceci est le même principe pour tous les axes, et si nous voulons aller en diagonale alors il suffit de modifier deux axes.

Dans notre cas, nous voulons qu’à chaque fois la main se retrouve avec la même position quel que soit la position de base que l’utilisateur ait faite. Pour ceci, nous allons définir le centre de la paume de la main comme étant le centre du repère. Cela revient juste à déplacer la main entière d’une certaine manière pour que la position du centre de la main soit de coordonnées . Pour ce faire, il faut alors tout simplement soustraire à chaque vecteurs position de la main les coordonnées sur chaque axe du centre de la paume de la main :

Avec les coordonnées d’un vecteur quelconque, étant ici l’un de ceux de la main, et les coordonnées du centre de la paume de la main.

En faisant cela, nous recentrons chaque main, quelle que soit sa position de base, vers le centre du repère.

Comme on vient de le voir, on peut le faire en utilisant un vecteur que l’on soustrait à chaque vecteur position de la main. Cependant, dans notre cas, nous n’allons pas utiliser cette technique, mais plutôt des matrices.

Une matrice est une sorte de tableau rectangulaire avec un certain nombre de lignes et de colonnes, où chaque élément est présent (aucune case n’est vide). Elle peut se représenter de cette manière :

Où est une matrice de taille , avec son nombre de lignes, et son nombre de colonnes.

Différentes opérations sont possibles entre des matrices, telles que la multiplication, l’addition et la soustraction, mais pas la division. L’addition et la soustraction sous soumis à la condition que les matrices incluses dans l’équation doivent avoir le même nombre de lignes et de colonnes. Ces deux opérations sont simples. Elles font, pour chaque élément à la même position dans les matrices, l’addition ou la soustraction de ceux-ci.

Où et

Pour le produit matriciel, c’est-à-dire la multiplication de matrices, ceci est différent. La condition est que, pour la multiplication entre deux matrices, il faut que le nombre de colonnes de la première soit égal au nombre de lignes de la deuxième. Ceci fait que si le produit matriciel est possible, cela ne veut pas dire que est possible. En effet, si on prend une matrice de taille , et B de taille , alors pour que soit possible il faut , et pour , il faut . De plus, même si et sont tous deux possibles, la plupart du temps .

Une fois le produit matriciel effectué, on obtient une matrice de taille avec le nombre de lignes de la première matrice, et le nombre de colonnes de la seconde. L’élément de cette matrice provient de la somme des multiplications des éléments de la ligne i de la première matrice, et des éléments de la ligne j de la seconde matrice. Pour que ce soit plus simple de compréhension, on va représenter la multiplication des matrices comme cela :



Où la matrice de gauche est , la matrice du haut est , et enfin, la matrice restante est la multiplication .

Le premier élément de la matrice résultat, c’est-à-dire à la position , s’obtient en faisant la somme des multiplications de la première ligne de la première matrice, par les éléments de la première colonne de la seconde matrice. Dit autrement, on fait la somme de pour x allant de à . Pour les autres éléments, on décale d’une ligne ou une colonne. Pour l’emplacement ce sera la somme de toujours pour x allant de à . Pour généraliser, on peut dire que si on cherche l’élément du produit matriciel, on peut le trouver en faisant :

Où là aussi , soit correspond au nombre de colonnes de la matrice , et au nombre de lignes de la matrice .

Rajouter un schéma pour plus de compréhension

Il existe également une matrice carrée (le nombre colonnes est le même que le nombre de lignes) nommée identité, qui possède sur sa diagonale (de gauche à droite vers le bas) des 1, et sur le reste de ses éléments des 0. Celle-ci a une propriété particulière. Pour toute matrice de taille correcte pour la multiplication, le produit matriciel renverra la même matrice. C’est-à-dire :

et

Pour I la matrice identité, et A une matrice pouvant être multiplié par I.

Comme vous pouvez vous en douter, pour qu’un vecteur et une matrice puissent être multipliables entre eux (matrice fois vecteur, par l’autre sens), il faut que la taille du vecteur soit égale au nombre de colonne de la matrice. Cependant dans notre cas, nous allons rendre le vecteur homogène. Ceci signifie qu’il possède une ligne en plus, et donc la matrice devra avoir une colonne supplémentaire.

Faire ceci permet en général d’avoir de biens meilleures performances dans les logiciels informatiques. Le principe est que pour avoir le vrai vecteur, il faudra diviser chacun de ses éléments, par le nouvel élément présent à la fin. En effet, si par exemple on a des nombres avec 5 chiffres après la virgule, alors pour faire les calculs, on les multipliera par , dans la mesure où l’ordinateur est plus puissant pour les calculs sans virgule. Ainsi, si on a plusieurs calculs à la suite sur ces mêmes nombres, on gagne en performance. Et lorsque l’on voudra le vrai vecteur, il faudra diviser tous les éléments par . Ainsi, la valeur du nouvel élément du vecteur sera dans ce cas-là.

Ensuite, pour la matrice, nous préférons utiliser une matrice carrée. Ceci nous sera utile pour les futurs calculs, lorsqu’on voudra multiplier plusieurs matrices entre elles, si elles sont toutes carrées et de même taille, nous n’aurons pas de problème de sens de produit matriciel. Par conséquent, nous utilisons toujours la matrice identité, mais avec une ligne et une colonne en plus (donc tous les nouveaux éléments sont à 0, sauf celui du coin inférieur droit, qui est à 1, car il est sur la diagonale).

Notre premier but était de faire une translation sur chaque vecteur de la main, c’est-à-dire retirer une certaine valeur en , une autre en , et de même en . Pour se faire, nous allons utiliser les coordonnées homogènes. Rien de bien compliqué là-dedans dans la mesure où on vous a déjà introduit toutes les notions nécessaires.

On a vu que la multiplication de notre vecteur par la matrice identité nous rendait notre vecteur. Voyons la décomposition :

On peut remarquer sur la dernière colonne, pour les lignes de , et , la multiplication . On voit alors qu’en modifiant la valeur de la dernière colonne de la matrice, on peut modifier cette valeur :

Ainsi, on peut facilement faire notre translation en changeant les valeurs de , et , avec par exemple des valeurs négatives. Par conséquent, dans notre cas nous mettrons les valeurs du vecteur position du centre de la paume de la main, multipliées par pour bien qu’elles soient soustraites et non additionnées.

Par la suite, nous voulons faire une mise à l’échelle. Ceci revient à multiplier chaque valeur de chaque vecteur par un même nombre. Par exemple, si nous voulons que la main soit deux fois plus grande, on la multiplie par 2, et si on veut qu’elle soit deux fois plus petite, on la multiplie par . Pour faire cela, on reprend tout simplement notre matrice identité que l’on multiplie par notre valeur (ou on remplace les de la diagonale par notre valeur). Et ensuite on multiplie nos vecteurs par la matrice que nous avons obtenue :

Dans la suite de notre cas, notre vecteur est celui qui est déjà translaté, soit . Donc nous aurons :

Soit :

L’ordre est important, il faut bien multiplier la matrice de mise à l’échelle par la gauche. Pour optimiser nos calculs, nous pouvons calculer le produit matriciel des deux matrices afin de ne pas le faire à chaque fois :

Dans notre cas, nous voulons que chaque main, quel que soit sa taille de base, ai la même proportion. Pour ce faire, nous allons définir une taille souhaitée à la fin de la mise à l’échelle. Nous allons prendre la taille d’un os comme outil de comparaison. L’os métacarpien du majeur est le plus grand de la main, ce qui permettra d’avoir plus de marge qu’avec une phalange distale par exemple. Ainsi, notre but est qu’à chaque fois que la mise à l’échelle est faite, l’os choisi aura une taille que l’on aura décidé au préalable. Dans notre cas, nous allons dire 75 (il n’y a pas de réelle raison, mais c’est une taille qui ne diffère pas beaucoup de nos mains). Ainsi, nous avons la taille de l’os en entrée (que l’on récupère auprès du LeapMotion), et la taille en sortie souhaitée, ici 75. Alors on cherche pour :

Avec la taille de l’os en entrée et la valeur recherchée pour obtenir 75 à partir de .

Ainsi, on a trouvé que notre valeur de mise à l’échelle est divisé par la taille de l’os en entrée.

Maintenant nous avons notre centrée et mise à l’échelle, cependant, si nous l’affichons, elle tourne toujours.

Insérer une suite d’image pour montrer qu’elle tourne ?

Il existe des matrices dites de rotation, qui permettent, comme leur nom l’indique, de faire une rotation d’un vecteur. Cette rotation se fait suivant un axe, qui peut être un vecteur choisi suivant la matrice. Dans notre cas il n’est pas nécessaire de connaître comment ont été trouvés ces matrices, nous avons juste besoin de savoir les utiliser et ce qu’elles font.

Voici les matrices de rotations respectivement autour des axes x, y et z :

, ,

Avec l’angle de rotation en radiant.

Les rotations opèrent ainsi :

*tourne l'axe vers l'axe :*



*tourne l'axe vers l'axe :*



*tourne l'axe vers l'axe :*



On peut remarquer que ces trois matrices font tourner les vecteurs suivant un certain axe, qui ne bouge pas. Nous pouvons remplacer cet axe par un vecteur unitaire (c’est-à-dire que

). Pour cela, nous utilisons cette formule :

Avec et .

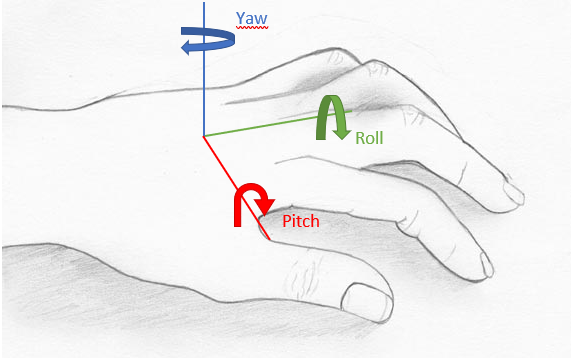
Le LeapMotion de son côté nous renvoie deux vecteurs unitaires et trois angles (en radians) de rotations pour chaque main. Les vecteurs sont ceux de la direction où pointe la main (donc vers les doigts), et celle où pointe la paume de la main.



Les rotations correspondent au yaw, pitch et roll (en anglais), soit respectivement la direction, l’aileron et la profondeur. Voici une image pour mieux comprendre :



Dans notre cas de la main, ceci donnerait plutôt :



C’est-à-dire que la rotation Yaw fait tourner la main avec le bras qui se déplace. Cette rotation se fait autour de l’axe du vecteur de la Paume. La rotation Roll fait lever ou baisser le pouce suivant l’angle. Elle se fait autour du vecteur Direction. Enfin, la dernière est Pitch, qui est quand on penche la main vers l’avant ou l’arrière. Elle correspond à un axe qui ne nous est pas donné par le LeapMotion.

Ainsi, toutes les rotations renvoyées sont dans le repère de la main et non dans celui du capteur LeapMotion. EXPLIQUER



Rotation x à 0 :



Rotation Y à 0 :



Rotation Z à 0 :









*Expliquer pk on ne fait pas les 3 matrices directement et plutôt on tourne déjà par rapport à un axe…*

*Dire aussi que le LeapMotion renvoie les vecteurs direction…*

*Et enfin de la dernière comparaison avec les matrices, normalisations et tout…*

**NE PAS SUPPRIMER : Faudra y déplacer, mais à la fin, pour le moment je m’en sers et ce n’est pas gênant d’y laisser là**

**Sources :**

[**https://developer-archive.leapmotion.com/documentation/v2/java/devguide/Leap\_Overview.html**](https://developer-archive.leapmotion.com/documentation/v2/java/devguide/Leap_Overview.html)

[**https://fr.wikipedia.org/wiki/R%C3%A8gle\_de\_trois**](https://fr.wikipedia.org/wiki/R%C3%A8gle_de_trois)

[**https://fr.wikipedia.org/wiki/Vecteur**](https://fr.wikipedia.org/wiki/Vecteur)

[**https://ent.uca.fr/moodle/pluginfile.php/1390368/mod\_resource/content/0/1a\_calcul\_matriciel\_slides.pdf**](https://ent.uca.fr/moodle/pluginfile.php/1390368/mod_resource/content/0/1a_calcul_matriciel_slides.pdf)

[**https://www.docteurclic.com/symptome/douleur-des-doigts.aspx**](https://www.docteurclic.com/symptome/douleur-des-doigts.aspx) **(Pour l’image)**

[**https://fr.wikipedia.org/wiki/Cosinus**](https://fr.wikipedia.org/wiki/Cosinus)

[**https://fr.wikipedia.org/wiki/Bijection\_r%C3%A9ciproque**](https://fr.wikipedia.org/wiki/Bijection_r%C3%A9ciproque)

[**https://fr.wikipedia.org/wiki/Arc\_cosinus**](https://fr.wikipedia.org/wiki/Arc_cosinus)

[**https://fr.wikipedia.org/wiki/Radian**](https://fr.wikipedia.org/wiki/Radian)

[**https://fr.wikipedia.org/wiki/Matrice\_(math%C3%A9matiques)#D%C3%A9finitions**](https://fr.wikipedia.org/wiki/Matrice_(math%C3%A9matiques)#D%C3%A9finitions)

[**https://fr.wikipedia.org/wiki/Produit\_matriciel**](https://fr.wikipedia.org/wiki/Produit_matriciel)

[**https://fr.wikipedia.org/wiki/Coordonn%C3%A9es\_homog%C3%A8nes**](https://fr.wikipedia.org/wiki/Coordonn%C3%A9es_homog%C3%A8nes)

[**https://fr.wikipedia.org/wiki/Matrice\_de\_rotation#Dimensions\_embo%C3%AEt%C3%A9es**](https://fr.wikipedia.org/wiki/Matrice_de_rotation#Dimensions_embo%C3%AEt%C3%A9es)

[**https://arduino103.blogspot.com/2015/08/detecter-les-mouvements-sur-le-hat-sense.html**](https://arduino103.blogspot.com/2015/08/detecter-les-mouvements-sur-le-hat-sense.html)

[**https://www.researchgate.net/figure/The-users-hand-with-the-three-axes-and-the-names-of-the-orientation-around-these-axes\_fig17\_46720588**](https://www.researchgate.net/figure/The-users-hand-with-the-three-axes-and-the-names-of-the-orientation-around-these-axes_fig17_46720588)

**TODO :**

**Pour expliquer le produit en croix plutôt prendre un exemple du projet, pas des champignons…**

**Pour diag de sequence, faire pour la méthode compareGesture…**

#### b. API

#### c. Base de données

Afin de permettre à l’utilisateur de stocker ces données en ligne et de pouvoir les récupérer depuis n’importe où nous avons mis en place une base de données. Nous voulions profiter de ce projet pour expérimenter l’interaction avec une base donnée NOSQL qui une famille de systèmes de gestion de base de données qui s'écarte du paradigme classique des bases relationnelles. En effet, comme le nom le suggère, celle-ci n’utilisent pas le langage bien connu SQL ou « Structured Query Language ».

Exemple d’un script stocké dans la base de données écrite en JSON :

{

"\_id": "ZWxweGV4YW50bA==",

"execType": "Python",

"args": ["-h"],

"file": "cGFyZW50cywgYmFiaWVzID0gKDEsIDEpCndoaWxlIGJhYmllcyA8IDEwMDoKICAgIHByaW5",

"description": "An Amazing script !!",

"idGesture": "b3BlbiBoYW5kYW4gb3BlbmVkIGhhbmQ=",

"\_class": "Core.Script.Script"

}

C’est pourquoi, nous avons choisi d’utiliser MongoDB. MongoDB est un système de gestion de base de données orienté documents, répartissable sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données. Ce choix a aussi été fait par rapport au Framework d’API Spring BOOT qui contient des méthodes déjà implémenté permettant d’utiliser MongoDB. De plus, cet outil est vraiment apprécié des développeurs et retrouvé dans un grand nombre de projets.

## V. Difficultés rencontrées

### 1.Difficultés techniques

### 2.Nouveau langage

### 3.Retards

## V. Bilan technique

### 1. Réalisations

### 2.Protocols expérimentaux

Un protocole expérimental a été réalisé lors de ce projet, celui-ci a pour but d’évalué la performance de notre algorithme de reconnaissance des gestes. Dans ce rapport se trouve le condensé du protocole expérimental réalisé. Ce protocole est récupérable en entier sur le git du projet dans le dossier conception.

Ci-dessous se trouve les résultats obtenus lors des expériences. Une expérience avec un geste simple tel que la pierre a été choisi, et une expérience avec un geste dit « compliqué » tel que le cœur fait avec deux mains a été choisi.  Tous deux ont pour but d’évaluer la reconnaissance d’un geste par rapport à un geste prédéfinis. Chaque expérience se déroule en cinq étapes. La première consiste à effectuer le geste enregistré le plus précisément possible pour savoir si l’algorithme reconnait avec ou sans difficulté le geste de base. Ensuite les quatre dernières étapes consistent à reproduire le geste enregistré avec différentes inclinaisons comme le montre les photos présente ci-dessous. Une batterie de cinquante gestes a été effectué pour chaque étape. Après avoir pris notes des résultats obtenu par l’algorithme ceux-ci ont été transformé en pourcentage que vous pouvez retrouver ci-dessous.



Tableau de résultats d’expérience :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Expérience | Gestes | Inclinaison à gauche | Inclinaison à droite | Inclinaison en avant | Inclinaison en arrière | Geste effectué avec précision |
| N°1 | Pierre | 90% | 72% | 78% | 82% | 100% |
| N°2 | Cœur | 0% | 0% | 0% | 0% | 82% |

*%=Taux de reconnaissance*

Ces expériences nous révèlent que notre algorithme est efficace si l’utilisateur reproduit avec le plus de précision possible son geste, 100% de reconnaissance pour la pierre et 82% pour le cœur. Seulement dès que le geste est reproduit avec différentes inclinaison le taux de reconnaissance baisse pour un geste simple et devient nul pour un geste compliqué comme le cœur.

## VI. Conclusion

### 1. Réalisation finale

Pour conclure, nous avons réussi à réaliser toutes les fonctionnalités promises dès l’établissement du cahier des charges.

### 2. Evolution du projet dans le futur

Le projet étant clôturé dans le cadre du DUT nous avons tout de même des idées pour le complété. Nous pensons que la plus grosse amélioration serait de supprimer toutes les dépendances entre le Back-end et le Front-end. Cela nous permettrait alors de totalement découpler les deux parties et ainsi de proposer à des utilisateurs une prise en charge des services de reconnaissance à distance permettant alors à la machine de l’utilisateur, potentiellement un mini-contrôleur, de seulement avoir à envoyer les images prise par le Leap Motion et à exécuter le script lorsqu’un des gestes est reconnu. Cette idée, a été proposé par Mr Bouhours et c’est dans cette ambition que nous avons choisi de faire communiquer le Back-End et le Front-End avec une API. Cependant c’est un objectif que nous n’avons pas pu atteindre faute de temps. Nous pensons que c’est la principale voie d’amélioration du projet et nous avons déjà mis des points d’extension permettant d’implémenter cela plutôt facilement.

## VII. Résumé en Anglais

This project is part of our two-year university diploma in Computer Science at the University Institute of Technology, UCA in Clermont-Ferrand. We did a four-month project in a group of 5.

The goal of the project is the development of software allowing the use of the Leap Motion device to launch scripts from predefined gestures which will then be recognized by the device. The Leap Motion is a computer hardware sensor device that supports hand and finger motions as input, analogous to a mouse, but requires no hand contact or touching. We used this camera and the library made available by the developers of the Leap Motion to develop this application.

In conclusion the project is up to what we wanted to do at least but it remains a lot of ideas that came up during the realization. If we had more time, we would have definitely disassociated the Back-End and the Front-End in order that this application would be totally optimized. But after all we are totally satisfied with what we managed to do in 14 weeks in a group of 5.

# BIBLIOGRAPHIE et WEBOGRAPHIE

# LEXIQUE

Leap Motion : capteur permettant de virtualiser nos mains. Cela nous permet ainsi de lancer un traitement prédéfini lorsqu'un certain mouvement est reconnu.

Back-End : terme désignant un étage de sortie d'un logiciel devant produire un résultat. C’est la partie invisible de l’iceberg qui permet de faire les traitements nécessaires afin d’échanger les données avec le Front-End.

Front-End : terme désignant toutes les parties d’un logiciel avec lequel un utilisateur va interagir. On retrouve ici la partie visible de l’iceberg (interface graphique, interface en invite de commande…)

API : ou « interface de programmation d’application » ou « interface de programmation applicative » est un ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.

# ANNEXES