BOUISSIERE Mathieu
MANJARRES BUSTOS Juan David
GONZALEZ-CONDE Damien
MIQUET Emmanuel

Projet Long

–

Amélioration des performances du sportif à l'aide d'un outil de reconnaissance de mouvements

8 février – 11 mars 2016

Responsables

Euriell Le Corronc

Elodie Chanthery

Sommaire

I. Introduction

Dans le monde du sport, la recherche de la performance, d'exploits, de nouveaux records ne connaît pas de frein. Mais en parallèle, la sureté médicale dans le domaine sportif est un enjeu toujours plus important car le corps humain est poussé dans ses retranchements, et on souhaite à la fois de la performance et de la sécurité, pour les sportifs de haut niveau, mais aussi pour le grand public.

Aussi, le domaine de l'interaction entre l'Homme et les machines informatiques cherche lui aussi à repousser les limites depuis quelques années avec l'apparition et le développement de technologies permettant un contrôle toujours plus intuitif des dispositifs.

Le jeu vidéo est un monde où la technologie peut s'exprimer à son plein potentiel. En particulier, dans le monde de la capture de mouvements, qui n'a fait que s'étendre depuis la sortie de l'Eye Toy conçu par Sony pour la Playstation 2 en 2003, et qui permettait déjà d'interagir avec les jeux à partir d'une simple caméra.

Depuis, les technologies ont évolué et permettent des projets plus ambitieux. C'est notamment le cas du capteur Kinect développé par Microsoft, sur lequel nous avons travaillé au cours de ce projet. L'idée est de tirer avantage des progrès réalisés dans la capture de mouvements pour analyser les mouvements de sportifs afin de prévenir les mauvaises pratiques, potentiellement dangereuses ou inefficaces.

Pour réaliser ce stage d'une durée de six semaines au sein du LAAS[1], nous étions une équipe de quatre personnes, tous issus du département Génie Électrique et Automatique de l'ENSEEIHT[2], et plus particulièrement du parcours Commande, Décision et Informatique des Système Critiques. Pour réaliser l'objectif du stage, cette période de projet a vu naître plusieurs étapes distinctes. Il s'agissait d'abord de faire un état de l'art des solutions logicielles disponibles pour traiter les données fournies par une caméra Kinect (développée par le géant de l'informatique Microsoft), puis d'utiliser ces données à des fins de reconnaissance de positions et de mouvements pour, à terme, être capable d'effectivement analyser les mouvements caractéristiques de sportifs. Nous avons travaillé dans les locaux et au sein de l'équipe DISCO[3], dédiée à l'étude des méthodes de diagnostic à partir de modèles de systèmes ou de traitement de données.

II. Cadre et gestion du projet

L'ensemble du projet a été tourné vers du développement logiciel, avec le développement successif d'applications pour récupérer et mettre en forme les données souhaitées, puis pour les traiter, c'est-à-dire l'implémentation d'algorithmes de diagnostic de positions et de mouvements.

Pour pouvoir effectuer ce développement dans les meilleures conditions possibles, et tout en s'inspirant des pratiques couramment utilisées dans le monde de l'informatique au niveau professionnel, nous nous sommes tournés vers une méthode de développement dite « agile », la méthode *scrum*, que nous avons mise en place en l'adaptant à la durée de l'expérience de stage et à la taille de l'équipe de développement. De manière générale, les méthodes qualifiées « d'agiles » se distinguent des méthodes de développement traditionnelles en reposant sur la réalisation d'une solution par étapes incrémentales

[1] Laboratoire d'Analyse et d'Architecture des Systèmes, membre du Centre National de Recherche Scientifique
[2] École Nationale Supérieure d'Électronique, d'Électrotechnique, d'Informatique, d'Hydraulique et des Télécommunications, membre de l'Institut National Polytechnique de Toulouse
[3] DIagnostic, Supervision et COnduite

simples mais fonctionnelles. L'objectif est évidemment un gain de rapidité, de souplesse et de réutilisation dans le code réalisé.

La méthode *scrum*, dont le nom est directement tiré de la traduction anglophone de la mêlée au rugby, est réellement apparue à la fin des années quatre-vingt-dix. Cette méthode se caractérise par un développement par « sprints », c'est-à-dire en périodes de deux à quatre semaines qui amènent à une itération, une version de la solution. Dans notre cadre d'un stage court de six semaines, nous avons plutôt visé des sprints d'une durée de dix jours à deux semaines environ, afin de pouvoir effectuer réellement plusieurs itérations. Au cours d'un sprint, la définition de l'objectif principal pour la période ne doit pas être modifiée. Une itération doit amener à une version de code qui fonctionne, que l'on aura testée et que l'on pourra réutiliser au maximum, soit comme « boîte noire », en tant que service, soit pour ajouter des fonctionnalités au code existant.

Une des notions clés des méthodes agiles, et donc de la méthode *scrum* en particulier, est la communication au sein de l'équipe. Toutes les parties prenantes du projet doivent avoir accès à tout moment à l'avancement du projet, à l'ensemble des tâches en cours, et ce afin d'accroître la réactivité en cas de nécessité d'ajustements, ou de divergence des développements. Pour inciter les membres de l'équipe à communiquer et pour faciliter cette connaissance globale de l'avancement du projet, la méthode encourage à effectuer de brèves réunions très fréquentes (quotidiennes idéalement), où chacun présente successivement ce qu'il a fait depuis la précédente réunion, ce qu'il prévoit de faire avant la prochaine, et les obstacles qu'il rencontre pour y parvenir. L'objectif de ces réunions est bien l'information générale, et non la résolution des différentes difficultés, qui peut survenir après, dans les périodes de développement à proprement parler. Dans notre cas, nous nous sommes efforcés de maintenir un rythme régulier de mise à jour de l'avancement de chacun de façon informelle, sans pour autant imposer des réunions quotidiennes compte tenu de la taille réduite de l'équipe et de la communication continue au cours des journées de travail.

En plus de ces mises à jour internes et régulières, chaque sprint se conclue par une « revue de sprint », c'est-à-dire par une présentation de la solution courante à l'ensemble de l'équipe, ainsi qu'aux responsables du projet, avec une ou des démonstrations, et des explications haut niveau, du moins dans un premier temps, des fonctionnalités implémentées. Ces revues se concluent par la planification de l'objectif pour le prochain sprint.

En parallèle de ces échanges, et pour se rapprocher encore de l'utilisation réelle de la méthode *scrum* qui encourage au découpage des travaux en tâches élémentaires, nous avons mis en place, surtout pour la deuxième partie de la période de stage, un système de tâches, sous forme d'un tableau avec la liste des tâches à réaliser, les personnes qui s'engagent à leur réalisation et leur état d'avancement. Cela constitue en fait un découpage plus fin de l'objectif de sprint et permet de garder à l'esprit des priorités et des objectifs à court terme clairs. Le tableau suivant regroupe l'ensemble des « macro-tâches » réalisées au cours de ce projet ainsi que les ressources humaines et le temps investis pour chacune.

Extraction des données		
Tache	**Date de réalisation**	**Acteurs**
Kinect **Compréhension du Fonctionnement & acquisition des données**	01/02 – 08/02	4
Formation au langage C#	01/02 – 14/02	1/2

Acquisition MATLAB et Interface Graphique (visuelle)	01/02 – 09/02	1
Acquisition C#, Interface Graphique & Extraction des points	01/02 – 14/02	3
Amélioration de l'interface C#, de la récupération des données et de leur mise en forme (fichier txt)	15/02 – 22/02	1/2
Plateforme de travail		
Tache	**Date de réalisation**	**Acteurs**
Compréhension et mise en place du GIT HUB	15/02 – 16/02	1/2
Classification des mouvements		
Tache	**Date de réalisation**	**Acteurs**
Etat de l'art des méthodes de classifications	09/02 – 14/02	2
Comparaison des méthodes de classification [MATLAB]	15/02 – 19/02	1
Classification de mouvements simples (méthode 1) [MATLAB]	20/02 – 25/02	1
Travail sur les Arbre de décisions	26/02 – 01/03	1
Classification de mouvements simples (méthode 2) [MATLAB]	20/02 – 01/03	1/2
Classification avec le software Orange	26/02 – 01/03	1
Présentation et Rapport		
Tache	**Date de réalisation**	**Acteurs**
Préparation du rapport, de l'abstract & oral	01/03 – 06/03	4

Enfin, pour faciliter le travail collaboratif sur un tel projet, nous avons utilisé plusieurs outils couramment utilisés dans le domaine du développement logiciel, et même plus largement dans le monde de l'ingénierie et du travail.

En premier lieu, nous avons ouvert un dossier Google Drive commun afin de pouvoir échanger des idées sur l'approche du projet, entre nous, et avec les personnes encadrant le projet. Cela permet en effet un accès distant et commun aux informations, l'échange d'avis via un système de commentaires, et la consignation de l'évolution de l'approche de la solution. Nous y avons aussi régulièrement déposé des documents pour expliquer le fonctionnement d'un service ou d'un sous-programme développé au reste de l'équipe.

Ensuite, nous avons mis en place un système de gestionnaire de version. En effet lorsque l'on développe une application logicielle, qui plus est en équipe, les modifications sont nombreuses, et peuvent rapidement devenir désordonnées et chaotiques si l'on ne met pas en place un tel système de *versioning* du code afin de garder trace de toute modification et de travailler conjointement sur les dernières versions des sources. Nous avons pour cela choisi d'utiliser GitHub, à partir de la deuxième semaine de développement, pour créer un dépôt commun en ligne depuis lequel chacun a pu cloner une image sur son installation. Après quelques essais et la rédaction d'un tutoriel pour une utilisation simple au sein de l'équipe, nous avons pu progressivement porter notre code sur GitHub et ainsi se maintenir tous à jour des dernières modifications au fur et à mesure. Bien entendu, un tel dispositif implique de ne synchroniser que du code « propre », c'est-à-dire commenté, écrit lisiblement, et le plus fonctionnel possible. Il ne faut donc partager que du code testé, ou tout du moins en cours de test.

III. Présentation du support matériel : Kinect

III.1. Présentation générale

Kinect, initialement connu sous le nom « Projet Natal », est un hardware développé par Microsoft en 2009. C'est un périphérique externe destiné aux consoles de jeux-vidéo Xbox360 permettant aux utilisateurs d'interagir avec leurs applications par reconnaissance de mouvements ou commande vocale. Microsoft le commercialisa le 5 Janvier 2011. Il fut vendu à 8 millions d'exemplaires, dont 1 million en seulement 10 jours, d'où son entrée dans le Guinness Book comme « l'accessoire high-tech le plus vendu dans un bref laps de temps ».

Malgré l'hostilité du géant de l'informatique à la portabilité la Kinect sur les ordinateurs, il se rétracta pour finalement proposer la version « Kinect Version 1 for Windows » en Février 2012. C'est d'ailleurs sur ce modèle que nous avons travaillé tout au long de notre projet.

III.2. Fonctionnement

- Capteurs de Kinect
  - Lentilles détectant la couleur et la profondeur,
  - Micros à reconnaissance vocale,
  - Capteurs motorisés pour suivre les déplacements rotatifs.

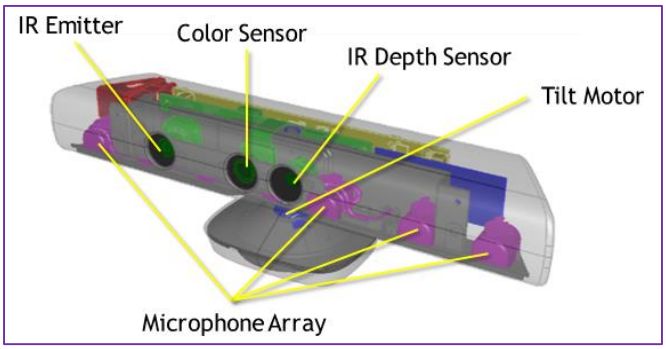

Figure 1:Capteurs de Kinect

- Caractéristiques du champ de vision
  - Angle de vision horizontal : 57,5°,
  - Angle de vision vertical : 43,5°,
  - Marge de déplacement du capteur : ± 27°.

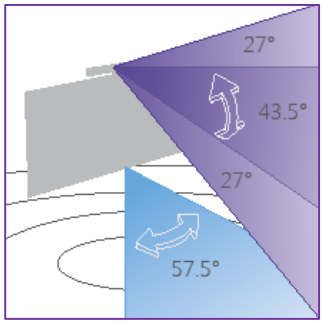

Figure 2:Champ de vision de Kinect

- Portée de capteurs de profondeur

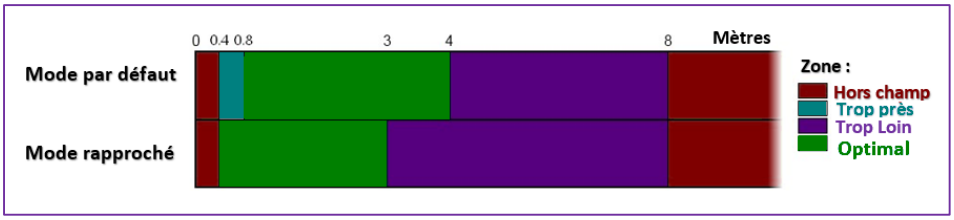

Figure 3:Portée des capteurs de profondeur

- Flux de données RGB et de profondeur disponibles
  - 320*240 pixels en couleur 16 bits à 30 images par secondes
  - 640*480 pixels en couleur 32 bits à 30 images par secondes

III.3. Reconnaissance des mouvements

Kinect peut détecter jusqu'à 6 personnes présentes dans son champ de vision. Comme illustré sur la figure suivante (*Figure 4*), le périphérique ne peut tracker (« suivre », de l'anglais *track*) que 2 personnes en mouvement. Les autres acteurs sont « non trackés », on connait juste leur « position globale ».

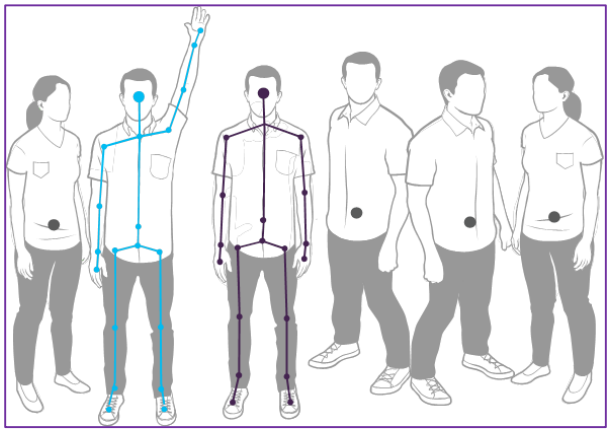

Figure 4:Plusieurs sujets dans le champ de vision

Le système de reconnaissance et de tracking de squelette peut être schématisé de la façon suivante (*Figure 5*) :

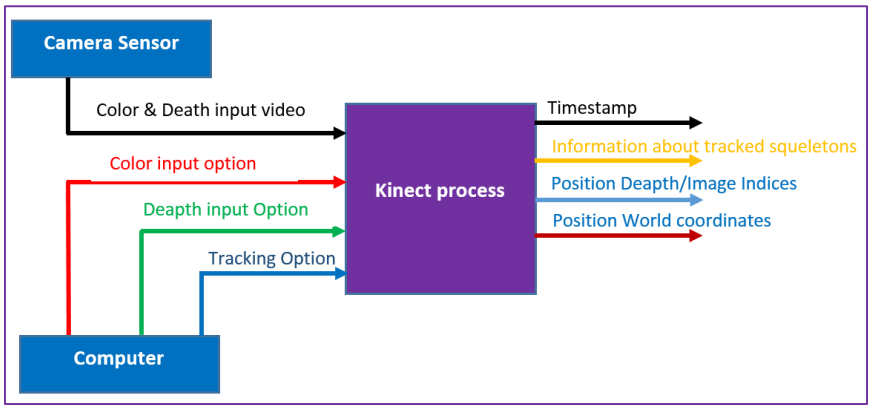

Figure 5:Schéma du système de tracking

En entrée du système, nous précisons les options de captures vidéo (nombre de frames par seconde, exposition, saturation …), les options de tracking (nombre de squelettes trackés, position assise ou levée …). Puis, nous récupérons des informations sur les positions des *Joints* (points caractéristiques) du/des squelettes.

Ces *Joints* correspondent pour la plupart, aux articulations réelles d'un être humain. Ils sont au nombre de 20. Nous verrons par la suite, que certains sont plus simples à tracker que d'autres !

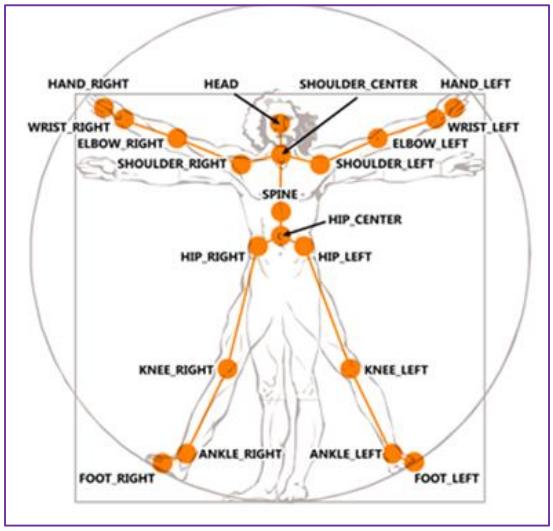

Figure 6:Répartition des points caractéristiques du squelette

Après avoir défini notre outil de travail, nous nous sommes concentrés sur le contrôle de la Kinect et sur l'extraction des données.

IV. Récupération des points à l'aide de Kinect

Pour que l'utilisateur puisse dialoguer avec Kinect, nous avons créé une interface permettant de visualiser le tracking des *Joints* et la récupération de leurs coordonnées. Les choix de programmation et les fonctionnalités de cette IHM, sont présentés dans cette section.

IV.1. Choix du système d'exploitation et du langage de programmation

Comme vu dans la présentation technologique de Kinect les seules informations brutes, les données de couleurs (capteur RBG) et les données de profondeur (émetteur/récepteur infrarouge). Ce sont les seules données auxquelles nous avons accès si nous n'utilisons pas les bibliothèques fournies par Microsoft (Software Development Kit).

De fait, développer avec un autre langage que le C++/C# ou sur une autre plateforme que Windows implique de se passer des avantages procurés par ces bibliothèques. En effet Kinect n'est pas seulement une caméra à laquelle on a ajouté un capteur infrarouge, elle est aussi capable de faire un traitement des données et donc de renvoyer d'autres informations comme la position des personnes ou leur nombre. En outre, le dispositif peut également repérer leur ''squelette'' et restituer les coordonnées des différentes articulations.

C’est pour ces avantages que nous avons décidé de coder notre application de récupération de points sur Windows et plus particulièrement en C#.

IV.2. Méthode de développement de l’application

Le développement de l’application de récupération de points a été pour nous l’occasion de découvrir un nouveau langage de programmation, le C#. En effet nous aurions pu coder en C++ mais le but de ce projet est aussi la découverte et l’élargissement de nos connaissances. De plus, en analysant différents codes pour Kinect nous avons bien vu que les fonctionnalités en C++ n’étaient que des portages des versions C# (donc moins lisibles sans connaissances particulières).

Nous avons abordé plusieurs méthodes quant à l’apprentissage du C#. Certains d’entre nous sont repartis de la base, en regardant des cours pour se former de façon plus traditionnelle. D’autres ont refait un programme de récupération de points minimal à l’aide de tutoriaux. Enfin, d’autres sont repartis de codes existants pour en comprendre le fonctionnement et ainsi en extraire ce qui nous intéressait. Ces différentes approches ont permis une montée en compétence plus rapide et partagée sur un langage qui nous était inconnu au départ de ce projet.

Une application commune a été développée à partir de ces différents travaux conjoints. Une fois cette application mise en place nous avons réduit les effectifs de développement (deux membres du groupe ont alors abordé la partie classification de notre projet).

Sur la suite du développement nous avons créé une bibliothèque de fonctions ‘’KinectTools’’ qui contient toutes nos méthodes, de l’acquisition des points, au traitement de ceux-ci. Cette bibliothèque a été conçue (comme toute bibliothèque) pour être portable à n’importe quel projet impliquant Kinect. La bibliothèque fait l’objet de la prochaine partie.

IV.2.a.Bibliothèque KinectTools

Cette boite à outils se compose de méthodes publiques que nous mettons à disposition du programmeur et de méthodes privées qui sont utilisées par notre bibliothèque elle-même et qui ne présentent donc pas d’intérêt pour le programmeur.

Le fonctionnement général de l’application est décomposable en deux temps : nous avons une méthode pour stocker les points dans des matrices quand ils arrivent (mode en ligne) puis un traitement (mode hors ligne) afin de nettoyer les données, calculer les angles et la longueur des membres.

Tous les fichiers générés sont faits pour être lisibles facilement par MATLAB, mais aussi dans un éditeur de texte (Notepad++ par exemple). Ils sont facilement transformables pour fonctionner avec d’autres logiciels de classification (logiciel R, Orange, …).

Nous allons maintenant nous intéresser plus particulièrement à certaines fonctions de cette bibliothèque afin d’en expliquer le fonctionnement. En voici la liste :

```
// Stocke les coordonnées des articulations voulues dans une matrice
public double joints2list(Skeleton skeleton, List<JointType> wantedJoints)

// Nettoie les données pour éliminer les erreurs
private void cleanSingularity(int nbPoints)

// Ecrit les coordonnées brutes et leur légende dans un fichier
public void joints2fileRAW(string TABpath, string jointsLegendPath, List<JointType>
wantedJoints)

// Ecrit les coordonnées traitées et leur légende dans un fichier
public void joints2file(string TABpath, string jointsLegendPath, List<JointType>
wantedJoints)

// Ecrit les coordonnées relatives (au centre du corps) et leur légende dans un fichier
public void relativeJoints2file(string TABpath, List<JointType> wantedJoints)

// Ecrit les angles voulus, les longueurs de membres et la légende dans un fichier
public void manageAngles(string anglesLegendPath, string bonesLengthPath, string
anglesDataPath, List<JointType> wantedJoints)

// Efface tout le contenu des listes temporaires et réinitialise le temps de capture
public void clearData()

// Permet d'obtenir la valeur RGB d'un pixel donné
public byte[] getRGBValue(ColorImageFrame imageFrame, int x, int y)

// Permet d'obtenir la valeur de profondeur d'un pixel donné
public int getDepthValue(DepthImageFrame imageFrame, int x, int y)

// Initialise la structure pour pouvoir s'occuper des angles
private void initWantedAngles(string anglesLegendPath, List<JointType> wantedJoints)

// Ecrit la longueur des membres dans un fichier
private void bonesLength(string bonesLengthPath, List<JointType> wantedJoints)

// Evalue la longueur des membres voulus (en moyenne sur une capture)
private double lengthEvaluation(JointType origin, JointType extremity, List<JointType>
wantedJoints)

// Evalue l'angle entre trois articulations
private double jointAngle(Joint middleJoint, Joint connectedJoint1, Joint
connectedJoint2)

// Evalue l'angle entre deux vecteurs
private double jointAngle(double[] vector1, double[]vector2)

// Effectue un produit scalaire
private double scalarProduct(double[] vector1, double[] vector2)

// Calcule la norme 2
private double norm2(double[] vector)
```

La méthode *CleanSingularity* a été premièrement développée sur MATLAB pour pallier aux problèmes d'acquisition. En effet, lorsque Kinect perd la trace d'une articulation il ne lui attribue plus de coordonnées. Nous lui attribuons alors (0,0,0) (pour les trois coordonnées spatiales X, Y, Z) pour signifier que le suivi a été perdu (un point n'atteint jamais naturellement les coordonnées nulles, donc les seuls points ayants ces coordonnées sont ceux dont Kinect a perdu la trace). Nous traitons alors les données stockées dans les matrices temporaires de la façon suivante :

```
private void cleanSingularity(int nbPoints)
{
    int ndt = 0, k = 0; //Initialisation des variables
    float dt = vect_t[1] - vect_t[0]; //Calcul du temps entre deux échantillons
    for (int i = 0 ; i < nbPoints ; i++) //On parcours toutes les articulations
    {
        for (int j = 0 ; j < Xi.Count ; j++) //Pour chaque instant
        {
            if (Xi[j][i] == 0) //Si la composante en X est nulle
            {
                for (k = j; k < Xi.Count; k++) //On cherche le prochain X != 0
                {
                    ndt++; //On incrémente le temps qui s'est écoulé
                    if (Xi[k][i] != 0) //Si on a trouvé une composante non nulle en X
                    {
                        break; //On sort de la boucle for
                    }
                }
                if (k != Xi.Count) //Si on a trouvé un X non nul en parcourant
                {
                    if (j == 0) //Si c'est le premier échantillon temporel
                    {
                        Xi[j][i] = Xi[k][i]; //X = premier X non nul
                    }
                    else //Pour tous les autres échantillons temporels
                    {
                        //Méthode du gradient sur tout l'intervalle où X est nul
                        Xi[j][i] = ((Xi[k][i]-Xi[j-1][i])/((ndt+1)*dt))*dt + Xi[j-1][i];
                    }
                }
                else //Si on a pas trouvé de X non nul en parcourant
                {
                    Xi[j][i] = Xi[j - 1][i]; //X = valeur du X précédent
                }
                ndt = 0; //Remise à zéro des variables
                k = 0;
            }
... //Même principe pour Y et Z dans la suite du code
```

Pour ce qui est de la méthode de calcul des angles entre les membres *jointAngles*, elle s'articule autour du principe simple suivant (issu de la définition du produit scalaire) :

$$\langle V_a | V_b \rangle = \|V_a\| \times \|V_b\| \times \cos(\theta_{ab})$$
$$\theta_{ab} = \operatorname{acos}\left(\frac{\langle V_a | V_b \rangle}{\|V_a\| \times \|V_b\|}\right)$$

Ainsi il suffit de calculer les vecteurs à partir des coordonnées des articulations et d'effectuer le calcul ci-dessus (avec les fonctions de normes et de produit scalaire que nous avons ajouté à notre bibliothèque) pour obtenir les angles entre les articulations souhaitées.

Toutes les fonctions sont disponibles dans KinectTools, donc le choix de l'utilisateur est indépendant du fonctionnement. Si l'on choisit de ne pas capturer le poignet, l'angle du coude ne sera pas calculé et

par conséquent tout ce qui le concerne ne sera pas écrit dans les fichiers. Tout ceci est effectué de façon transparente pour l'utilisateur et le programmeur.

La méthode *relativeJoints2file* permet d'éliminer l'influence de la proximité et/ou de la position du sujet par rapport à Kinect. Cette démarche est utile pour le travail de classification qui est abordé plus loin dans ce rapport.

IV.2.b.Application « Skeleton Tracking »

La bibliothèque décrite précédemment est utilisée dans le cadre d'une application de capture complète développée par nos soins. La version la plus avancée de l'interface graphique de l'application développée est illustrée à la figure suivante (*Figure 7*).

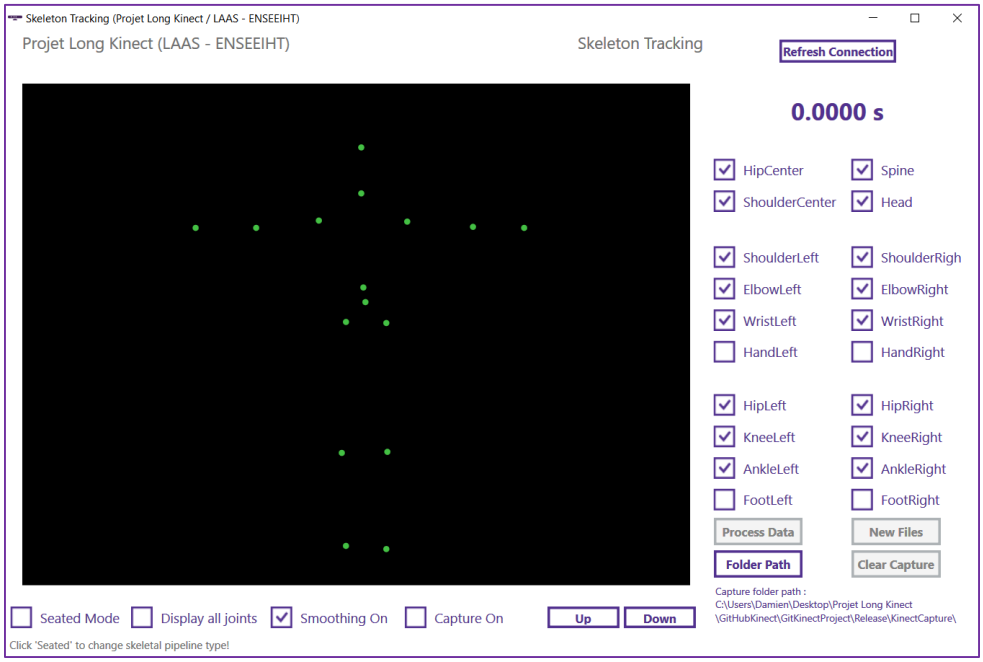

Figure 7:Interface graphique développée en C#

Elle se décompose en plusieurs parties. Sur la gauche se trouve une fenêtre de visualisation où l'on peut observer les articulations du squelette. Il aurait été possible d'afficher les articulations par-dessus l'image RGB et ainsi voir observer la superposition. Cependant, dans le but d'optimiser la rapidité de traitement (si l'on demande les deux images à Kinect, on réduit la fréquence des images), nous avons préféré opter pour la visualisation du squelette seul.

Cette visualisation est néanmoins modulaire. Au travers des différentes cases à cocher sur la droite, il est possible de choisir quelles articulations suivre. Cela impacte la visualisation mais aussi les fichiers qui sont générés par la suite. De plus, il est possible de changer le mode de visualisation en *Seated Mode*, ce qui permet de ne prendre en compte que le haut du corps. ''Display all joints'' permet de visualiser toutes les articulations quel que soit le choix fait avec les cases à cocher.

Smoothing On permet d'appliquer une correction sur les données brutes de Kinect et ainsi lisser les mouvements (nous avons appliqué des paramètres permettant de lisser sans trop de latence).

Up et *Down* permettent de changer l'angle de la camera.

Capture On permet de lancer la capture. Ce faisant, le temps en haut à droite défile et il devient impossible de changer les articulations observées ou les paramètres de Kinect pour éviter de fausser les mesures.

Une fois la capture effectuée il est possible d'inscrire les données dans un ensemble de fichiers (*Process Data*) ou d'annuler la capture (*Clear Capture*). Si l'on veut prendre un nouvel ensemble de capture il suffit d'appuyer sur ''New Files'' (il est aussi possible de choisir le chemin d'enregistrement avec *Folder Path*).

Quand toutes les captures souhaitées ont été faites on obtient les ensembles de fichiers suivants :

- i_anglesData.txt : Valeurs des angles à chaque instant

```
%             ShoulderCenter        ...
%             Head                  ...
%             Spine                 ...
% Time        Theta (Deg)           ...
0.00000000    179.537580450885      ...
0.03333334    179.285852212862      ...
...           ...                   ...
```

- i_anglesLegend.txt : Légende des angles suivis

```
ShoulderCenter - Head - Spine
Spine - ShoulderCenter - HipCenter
ElbowRight - WristRight - ShoulderRight
...
```

- i_bonesLength.txt : Longueur des membres

```
ShoulderCenter - Head – Spine       ShoulderCenter – Head       0.183890074396155
                                    ShoulderCenter – Spine      0.375189588084771
Spine - ShoulderCenter - HipCenter  Spine – ShoulderCenter      0.375189588084771
                                    Spine – HipCenter           0.0865086614249525
...                                 ...                         ...
```

- i_pointsCapture.txt/i_relativePointsCapture .txt : Valeurs de positions absolues/relatives

```
%Time         HipCenter-X          HipCenter-Y          HipCenter-Z    ...
0.00000000    0.00638657           -0.06402658          -0.06017685    ...
0.03333334    0.00639497           -0.06395509          -0.06026530    ...
0.06666667    0.00633386           -0.06380074          -0.06045318    ...
...           ...                  ...                  ...            ...
```

- i_jointsLegend.txt : Légende des articulations suivies

```
HipCenter
Spine
ShoulderCenter
Head
...
```

IV.3. Une autre possibilité avec Matlab

Au début de la période de stage, lorsque nous avons exploré plusieurs solutions de récupération de points, nous avons en particulier développé les prémisses d'une autre interface graphique utilisant les bibliothèques proposées par Matlab pour interagir avec des dispositifs d'acquisition d'image tel que Kinect. Celle-ci est capable d'afficher les *Joints* et le squelette retracé à partir du tracking issu des flux de données de couleur et de profondeur.

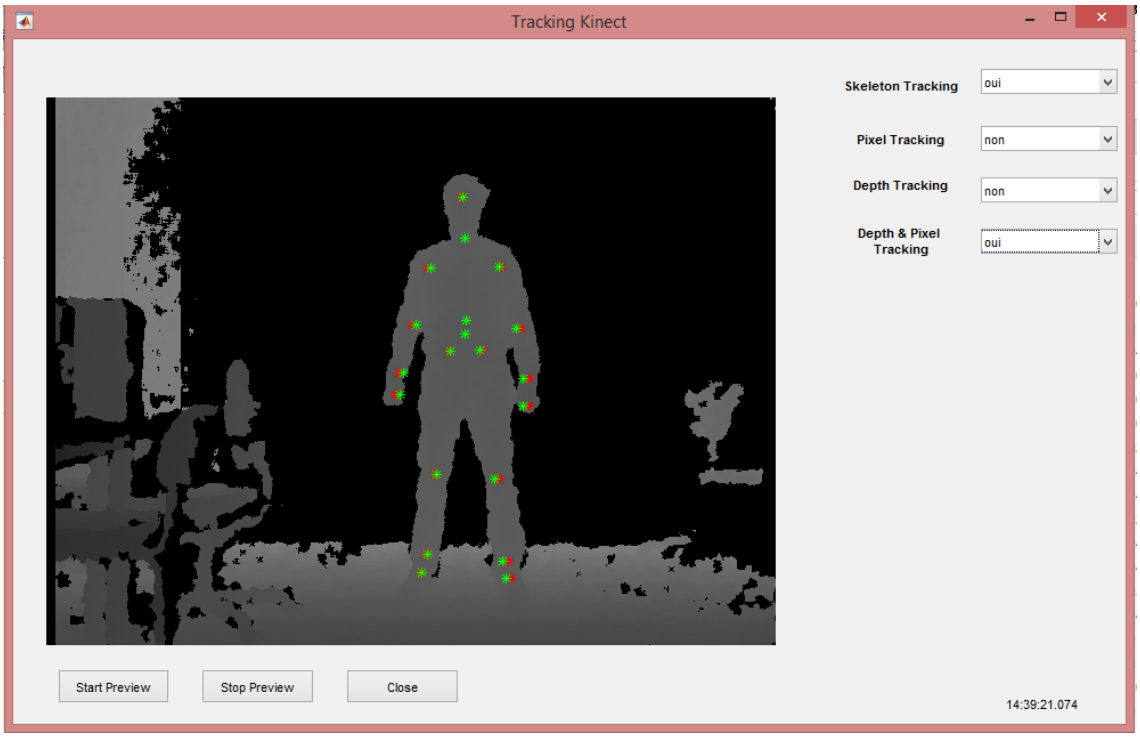

Figure 8:Interface graphique développée avec Matlab

Nous constatons lors d'essais que le tracking par les données du capteur RGB (points rouges) est moins précis que le tracking issu des données de profondeur (capteur infrarouge, points verts). Aussi, nous avons pu vérifier que l'infrarouge n'est pas perturbé par la qualité de l'éclairage ambiant, même en environnement sombre.

V. Mise en place de méthodes de diagnostic de positions

V.1. Principe général

De nos jours, du fait de la multiplication des capteurs et de l'amélioration des capacités de stockage, le traitement de grands jeux de données devient un enjeu crucial pour de nombreux marché industriels, ainsi que pour de nombreux domaines de recherche. En effet, si nous sommes capables de générer de grandes bases de données brutes, faire apparaître des corrélations entre ces données, afin de faire de l'analyse, voire de la prédiction, est indispensable pour leur donner un sens tangible et de créer des modèles réalistes. On parle de *big data* en anglais.

Dans notre cas, nous avons vu que nous sommes effectivement capables de recueillir les données de positions dans l'espace d'un certain nombre de *Joints* (points caractéristiques du squelette), ainsi que les valeurs des angles d'articulations. De plus, en tenant compte de la fréquence d'échantillonnage de d'échantillonnage de 30Hz environ (donc un jeu de données toutes les 33 ms d'acquisition), et du fait que pour effectuer des analyses pertinentes, il faudra se baser sur de nombreux essais, nous générons très rapidement des quantités de données conséquentes.

Une analyse sur de grandes quantités d'informations fait en particulier appel aux notions d'exploration de données (*data mining* en anglais) et d'apprentissage automatique ou statistique (*machine learning* en anglais). La terminologie dans ce domaine peut s'avérer être ambigüe, en anglais comme en français. Il faut donc définir ce que l'on entend par ces termes dans le cadre de notre étude.

De manière générale, ces procédés consistent en au moins deux parties distinctes : une phase d'apprentissage et une phase de prédiction. L'apprentissage consiste en l'entrainement d'une méthode à partir d'un ensemble de données témoin, et la prédiction a pour objectif de rapprocher de nouveaux jeux de données de situations, de connaissances issues d'un apprentissage préalable. Il existe deux types d'apprentissage : l'apprentissage supervisé et l'apprentissage non supervisé.

L'apprentissage supervisé requiert l'intervention d'un expert et une phase d'apprentissage hors ligne. En effet, il s'agit d'étiqueter manuellement les données afin de construire un modèle à l'aide d'algorithme. Le modèle peut ensuite être appliqué en ligne à de nouvelles instances de données pour leur attribuer l'étiquette la plus probable. Il existe plusieurs algorithmes différents pour effectuer de la classification supervisée comme nous le verrons dans le prochain paragraphe.

La seconde approche, dite non supervisée, ne requiert pas d'intervention extérieure, les algorithmes doivent partitionner les données sans « aide » (on parle alors de *clustering* en anglais), et l'opérateur devra par contre analyser le résultat de cette séparation des données, associer les groupes, leur donner du sens… On parle alors plutôt de diagnostic, lorsque l'on doit interpréter ces résultats, et apporter un regard critique, une prise de recul. Là aussi, des outils logiciels ou mathématiques peuvent aider l'expert.

Toutefois, ces deux approches ne sont pas complètement cloisonnées, et il existe des méthodes hybrides, semi-supervisées, ou par exemple des méthodes non supervisées pour lesquelles on peut spécifier des contraintes (nombre de groupements…).

Par la suite, nous étudierons les implémentations de différentes méthodes testées, leur fonctionnement spécifique, et les résultats obtenus.

V.2. Comparaison de différentes approches de classification supervisée

L'objectif de cette partie est de se familiariser avec plusieurs méthodes de classification supervisées, sous l'environnement Matlab, et d'en comparer les intérêts : simplicité de mise en œuvre, performances, robustesse…

Le logiciel de traitement mathématique Matlab propose plusieurs méthodes de classification :

- *Linear Regression*
- *Non Linear Regression*
- *Classification Trees*
- *Support Vector Machines*

- *Naive Bayes Classification*
- *Nearest Neighbors*
- *Model Building Assessments*

Après de brèves recherches sur ces différentes méthodes, nous avons décidé de comparer trois méthodes en particulier : *Classification Trees*, *Naive Bayes* and *Nearest Neighbors*, dans le but d'en connaître les avantages et inconvénients pour différentes positions du corps humain.

Nous avons aussi souhaité tester la méthode *Support Vector Machine*, mais le problème que nous avons rencontré est que cette méthode ne permet que la classification de jeu de données en visant deux classes. C'est pour cela que celle-ci n'a pas été retenue pour la suite car nous avions besoin d'un classificateur avec la capacité de distinguer plus de classes.

Nous avons divisé les tests en deux parties. Le premier test correspond à un essai sur deux classes, et le second test correspond à un essai pour trois classes.

V.2.a.Premier test : influence de l'entrainement (apprentissage)

Pour ce premier test, nous avons limité notre étude aux articulations suivantes :

- *Spine*
- *Shoulder Center*
- *Shoulder Right*
- *Elbow Right*
- *Wrist Right*

Soient deux classes pour le classificateur. Une bonne et une mauvaise position. On utilise un nombre de points identique pour l'entrainement de chaque classe. Le but de ce test est de comparer les différents résultats des classificateurs avec le même nombre de points pour l'entraînement de chaque classe. Les résultats sont les suivants (*Figure 9*) :

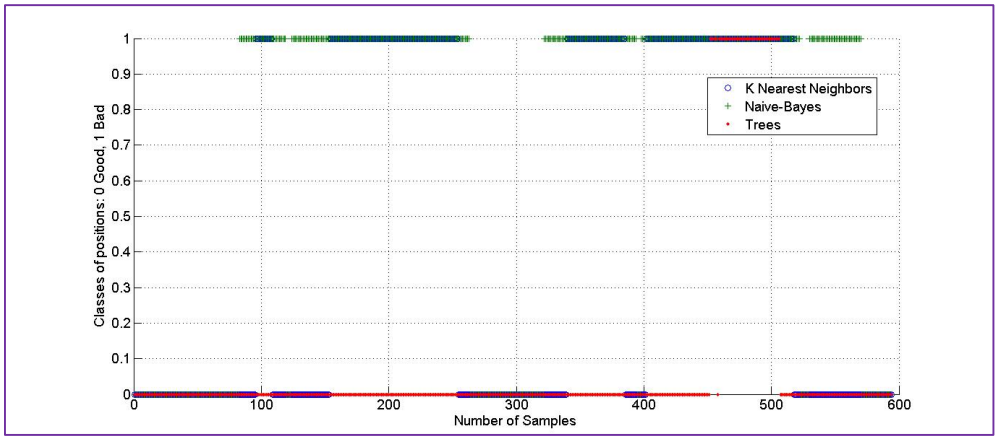

Figure 9:Comparaison des classificateurs, même entrainement et même groupe de test

Pour savoir quel classificateur donne les meilleurs résultats le groupe de données de test est constitué de bonnes et de mauvaises positions entrelacées. Le graphe attendu serait une alternance de la classe 0 (bonne position) et de la classe 1 (mauvaise position). On établit alors une comparaison à partir du graphe précédent et des résultats attendus.

Le tableau suivant présente cette comparaison :

Méthode	Résultat
Trees	Très loin de la réalité
Naive Bayes	Plus proche mais non satisfaisant
Nearest Neighbors	Proche de la réalité

Nous avons effectué un autre test, en faisant l'entrainement du classificateur avec plus de points pour une classe que pour l'autre. Le résultat est que naturellement, si on utilise plus de point pour une classe que pour l'autre, le classificateur est plus sensible à trouver la classe avec le plus de données.

V.2.b.Deuxième test : robustesse du classificateur au changement de sujet

Maintenant, on utilise trois classes qui correspondent à trois positions différentes. Les articulations monitorées sont les suivantes :

- *Spine*
- *Shoulder Left*
- *Shoulder Right*
- *Elbow Right*
- *Elbow Left*
- *Wrist Right*
- *Wrist Left*
- *Knee Right*
- *Ankle Right*

Le but de ce test est de savoir si les classificateurs sont robustes au changement de sujet de test. Le premier cas correspond à l'entrainement et au test avec la même personne pour les trois classes. Le deuxième utilisera le même entrainement mais avec une personne de taille différente pour le test.

Pour le premier cas, correspondant à l'entrainement et au test avec la même personne, l'entrainement a été fait avec des groupes de 20 points de chaque position. Pour faire le test la personne a fait les trois positions 3 fois. La première fois au centre du capteur, la seconde à droite, et la troisième à gauche mais avec une légère rotation autour de son axe de corps. On obtient les résultats suivants (*Figure 10*) :

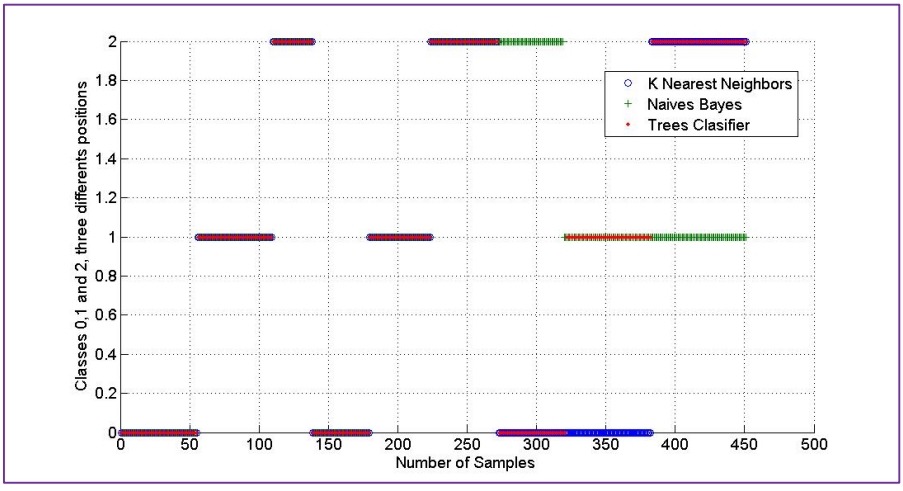

Figure 10:Comparaison des classificateurs, entrainement et test avec la même personne

Nous pouvons observer que les trois dernières positions (qui correspondent à la personne à gauche avec un peu de rotation autour de son axe) ont seulement été bien classifiées avec la méthode de Trees.

Avec le même entrainement que pour le cas 1, pour les trois classes, nous avons fait un nouveau test avec une personne de taille différente pour savoir quel est le classificateur le plus robuste au changement de ce paramètre. Dans ce cas nous allons comparer les résultats avec les classes de référence (*Figure 11*).

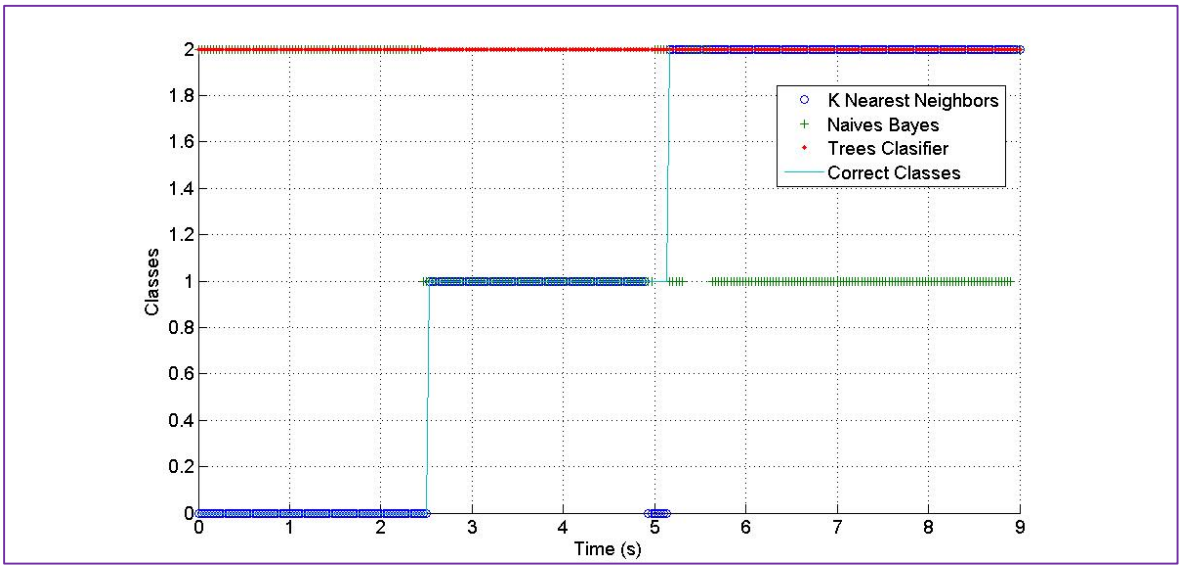

Figure 11:Comparaison des classificateurs, entrainement et test avec des personnes de taille différente

Le tableau suivant est un résumé des résultats illustrés par le graphique précédent :

Méthode	Résultat
Trees	Très loin de la réalité
Naive Bayes	Plus proche mais non satisfaisant
Nearest Neighbors	Quasiment parfait

Afin de mieux justifier notre choix de classificateur nous comparons aussi leur temps d'exécution.

Analyse du temps d'exécution	Nearest Neighbors	Naive Bayes	Trees
Temps d'exécution (s)	0.3052	0.1246	0.1084
Pourcentage de bonnes classifications (%)	97.4170	31.7343	42.8044

V.2.c.Conclusion sur les comparaisons de méthodes

Avec tous ces résultats nous pouvons conclure que le meilleur classificateur est le *Trees Classifier*, dans les cas où l'apprentissage et le test sont fait par la même personne.

Mais si nous changeons maintenant la taille de la personne, le meilleur classificateur est la méthode de *Nearest Neighbors*, elle est plus robuste aux changements, mais elle est aussi la méthode qui prends plus de temps pour faire son entrainement. L'important pour notre usage est surtout la robustesse, puisque les données de Kinect peuvent varier même pour une même personne. Le traitement étant fait hors ligne, le temps d'exécution à peine supérieur n'est pas vraiment un problème.

Finalement, pour réaliser des essais plus en profondeur par la suite en classification supervisée, nous avons choisi la méthode de Nearest Neighbors pour les raisons suivantes : La méthode est robuste aux changements de sujet d'expérience si l'entrainement est adéquat. De plus, même si elle est plus lente à entrainer elle donne de bons résultats de classification et elle est plus facilement maîtrisable.

Par la suite, nous présenterons toutefois d'abord une approche en classification non supervisée par la méthode dite des *K-Means* testée au cours de cette période de stage.

V.3. Les différentes implémentations effectuées sous Matlab

V.3.a. Implémentation d'une méthode de classification non supervisée pour un point du squelette

Dans un premier temps, nous avons étudié un mouvement simple, où le seul tracking d'un point caractéristique du squelette pouvait caractériser les différentes positions qui le composaient. Pour le mouvement présenté ci-dessous, nous avons pris le poignet comme point caractéristique. Le mouvement étudié est représenté dans la figure ci-dessous (*Figure 12*).

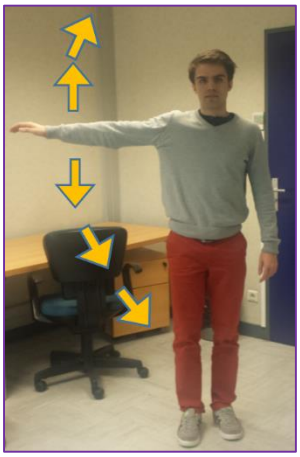

Figure 12:Mouvement étudié pour la première classification

Nous avons utilisé la méthode de classification de type « K-Means » pour différencier 5 positions dans le mouvement. L'idée est de diviser nos points en 5 partitions en minimisant un critère de distance (euclidienne dans notre cas). Chaque partition contenant un point central, l'objectif est de classer chaque donnée de façon à minimiser la somme des distances entre le point central (recalculé à chaque fois qu'un nouveau point est ajouté à la classe) et tous les autres points d'une classe. Notre programme est une heuristique dans la mesure où le résultat obtenu n'est pas optimal et qu'il dépend de choix arbitraires, comme la façon dont on initialise les points centraux (les barycentres).

La figure suivante (*Figure 13*) présente la localisation spatiale des *Joints* lors de l'exécution du mouvement, ainsi que l'attribution des classes (en couleur), et les barycentres correspondants.

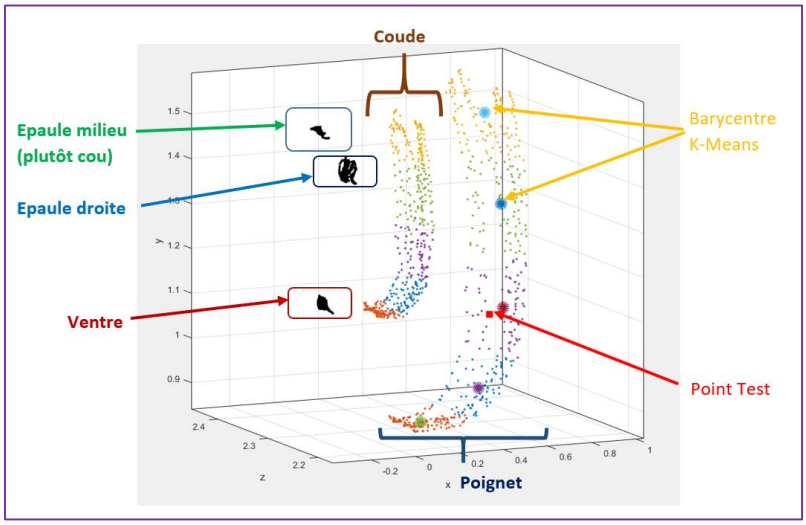

Figure 13:Représentation en trois dimensions des classes générées

Ensuite, nous avons pris les coordonnées d'une nouvelle instance, de test, de notre poignet, et nous avons cherché à utiliser l'algorithme sur une itération supplémentaire, de prédiction cette fois, pour connaître la classe à laquelle il associerait le nouveau point. Cet algorithme de prédiction calcule dans un premier temps la distance euclidienne entre le point central de chaque classe et le point test, puis garde celui avec la distance minimale.

Dans le cas où le mouvement est mal effectué, un autre programme vérifie des joints clés de positions tels que l'alignement de la colonne ou des bras.

Cette méthode est assez simple puisque le *K-Means* est une méthode non supervisée, mais elle comporte plusieurs défauts :

- La classification d'un seul type de joint n'est pas viable pour des mouvements complexes,
- Le programme de vérification de mouvement doit être revu pour chaque nouveau mouvement et même pour chacune des classes le composant,
- Le tracking de positions est affecté par la morphologie du sujet de test. Si la classification a été réalisée avec une grande personne, les joints d'une petite personne risquent de ne pas être mis dans les bonnes classes.

Ainsi, pour améliorer cette méthode, nous nous sommes par la suite focalisés sur le suivi des angles des articulations.

V.3.b.Classification par suivi d'angles

Nous reprenons le même mouvement que précédemment, mais en traitant les variations de certains angles, calculés à partir des positions des *Joints* pendant le mouvement enregistré (comme détaillé dans la partie précédente). Nous prendrons en particulier l'angle de l'épaule droite (entre les points épaule gauche - épaule droite - coude droit) comme sujet pour le classificateur. Nous choisissons d'imposer 10

centres (donc 10 classes) pour la méthode *K-Means*. Nous afficherons les valeurs d'autres angles pour vérifier la pertinence de nos résultats.

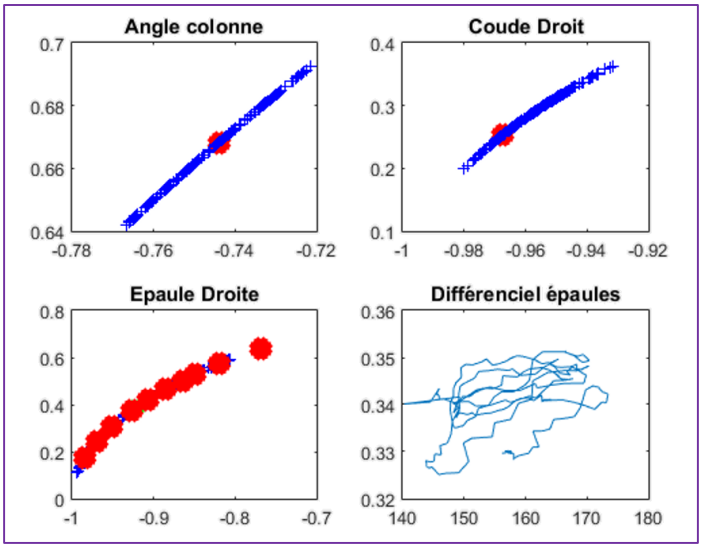

Figure 14:Résultats de la classification par suivi d'angle

Les angles ont été affichés sur un cercle trigonométrique de rayon unitaire. Les points rouges correspondent aux barycentres des classes. Le graphique « Épaule droite » de la figure suivante permet de suivre le mouvement du bras ; les trois autres courbes nous ont servis à vérifier l'exactitude de la qualité du mouvement.

Toutefois, nous observons que l'angle de l'épaule ne varie qu'entre les valeurs 0° et 180° (ou 0 et π radians). En effet, dans l'espace, les angles tels que nous les avons définis à partir du produit scalaire ne peuvent donner que l'angle convexe (le plus petit) entre deux vecteurs. Dans le cadre de ce mouvement spécifique, nous avons donc un problème de déterminisme, puisque les angles seront les mêmes pour plusieurs positions du bras.

Pour pallier à ce problème, nous avons étudié les angles susceptibles de varier au-delà de ces bornes, et nous avons décidé d'utiliser la projection des vecteurs correspondants dans un plan de référence (« plan frontal ») comme montré sur la figure suivante (*Figure 15*).

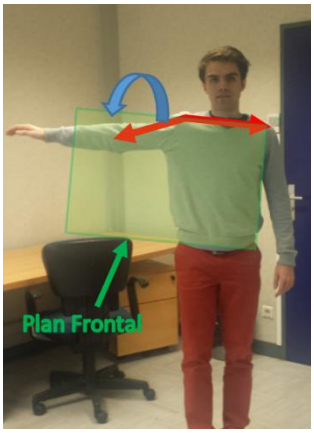

Figure 15:Projection des vecteurs dans le "plan frontal"

Nous allons désormais pouvoir exploiter les propriétés du produit, vectoriel cette fois, entre deux vecteurs afin de pouvoir caractériser des angles entre 0° et 360°. Ces transformations angulaires nous donnent de nouveaux résultats, présentés dans la figure suivante (*Figure 16*).

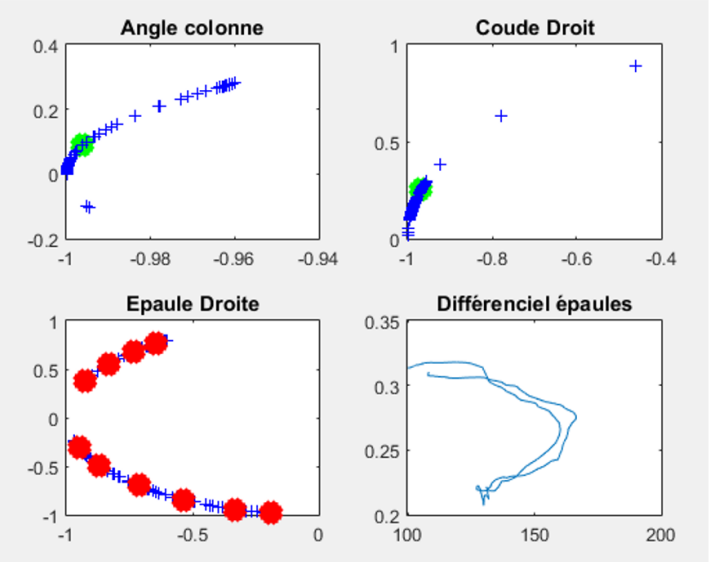

Figure 16:Résultats de la classification par suivi d'angle améliorée

La classification de notre mouvement est maintenant correcte et déterministe. De plus, les graphes des autres angles suivis (angle colonne correspondant au tronc de l'individu, et angle du coude droit) sont aussi beaucoup plus cohérents avec des valeurs proches de 180°, correspondant bien à un individu droit et effectuant le mouvement avec le bras tendu.

Ces premiers essais dans le monde de la classification nous ont permis d'en appréhender les principes fondamentaux, mais l'étude au travers des angles seuls montre que cette approche n'est pas adaptée à des différences de morphologie entre différents sujets, et même entre plusieurs essais. L'utilisation de données complémentaires semble donc indispensable à une étude vraiment fiable, et nous verrons l'importance du choix des jeux de données dans le paragraphe suivant sur notre implémentation de la méthode supervisée du *K-Nearest Neighbors* sous Matlab.

V.3.c.Adaptation de l'algorithme des *K Nearest Neighbors* (KNN)

Notre objectif est double pour cette partie. En effet il faut classifier les bonnes positions des sujets mais aussi rejeter les mauvaises. Il y a donc un aspect diagnostic au classificateur que nous allons mettre en place. Pour ce double but nous nous sommes basés sur la méthode des Nearest Neighbors. Le principe de cette méthode est, à partir d'un échantillon de test, d'évaluer la distance par rapport à une base de données de points référencés et d'y trouver le ou les points le(s) plus proche(s). Une fois que le ou les plus proche(s) voisin(s) dans la base de données sont déterminés, il est aisé de rattacher les points de tests à des classes que nous avons établies précédemment.

La méthode KNN disponible sur MATLAB ne communique pas les distances, mais seulement les classes prédites. Pour notre objectif nous avons besoin de cette distance afin de rejeter les points qui sont trop éloignés des classes souhaitées. En cherchant les fonctions utilisées par l'apprentissage de KNN nous avons trouvé une méthode qui, à partir des données d'apprentissage et des données de test, retourne la distance ainsi que la position (dans la matrice d'apprentissage) du ou des plus proche(s) voisin(s). À partir de ces données nous avons pu rejeter les points qui étaient trop loin de leur(s) plus proche(s) voisin(s) en utilisant une valeur de seuil. Lorsque la distance est en dessous de ce seuil, le point est classé dans la classe qui correspond à celle de son plus proche voisin dans la base de données. Dans le cas contraire ce point est mis dans une classe externe, donc rejeté.

```
function [classes,IDX,distance,threshold] =
KinectClassificationKNN(learningData,learnedClasses,testData,epsilon,K)
    [IDX,distance] = knnsearch(learningData,testData,'K',K);
    classes = zeros(length(testData(:,1)),1); % Initialisation des variables locales
    tmpClasses = zeros(1,K);
    d = zeros(1,K);
    threshold = min(min(distance)) + epsilon;

    if threshold >= 0.6*max(max(distance)) % Adaptation du seuil pour eviter les abus
        threshold = 0.6*max(max(distance));
    elseif max(max(distance)) < 1
        threshold = threshold/1.7;
    end
    for i = 1:length(testData(:,1)) % Pour tous les echantillons temporel
        for j = 1:K % Pour tous les voisins
            tmpClasses(j) = learnedClasses(IDX(i,j));
            d(j) = meanDistClass(tmpClasses(j),tmpClasses,distance(i,:));
        end

        [d , class] = min(d); % Trouver la classe dont la distance est la minimale

        if d < threshold % Si on est en dessous du seuil
            classes(i,1) = tmpClasses(class);
        else % Si on est au dessus du seuil
            classes(i,1) = 0;
        end
    end
end
```

Le code présenté ci-dessus est celui de la fonction que nous avons développé pour exploiter la donnée de distance. Il est possible de modifier le nombre de voisins souhaités grâce à la valeur K. Dans ce cas la distance prise en compte est la distance moyenne minimale par rapport aux différentes classes. Pour ce faire nous avons codé la fonction *meanDistClass* qui permet d'avoir la distance moyenne par rapport à une classe.

Afin d'utiliser ce classificateur de façon efficace, il faut normaliser les vecteurs de positions pour éliminer le plus possible l'influence de la taille du sujet. De plus nous utilisons les données des positions relatives des articulations (par rapport au centre du corps) pour s'affranchir de l'emplacement du sujet pendant la capture. Nous ajoutons à celles-ci les données angulaires, qui elles ne sont pas normées et donc ne créent pas d'interférences dans la classification. La concaténation de ces deux données de nature différente passe par une mise à l'échelle. En effet l'une est une donnée angulaire (ici en degrés) et l'autre

une donnée de position (en mètres). Nous divisons donc chaque donnée par leur maximum respectif afin de les normaliser entre zéro et un.

Nous allons maintenant présenter les résultats obtenus grâce à notre fonction. L'apprentissage est fait sur une première personne avec 3 positions (les bras tendus sur le côté, bras droit levé et bras gauche baissé, puis l'inverse). Le test est ensuite effectué avec une personne de taille différente. Une partie du test se compose de la succession des bonnes positions, puis une partie de positions aléatoires, et enfin deux bonnes positions. Les résultats sont disponibles ci-après (*Figure 17*).

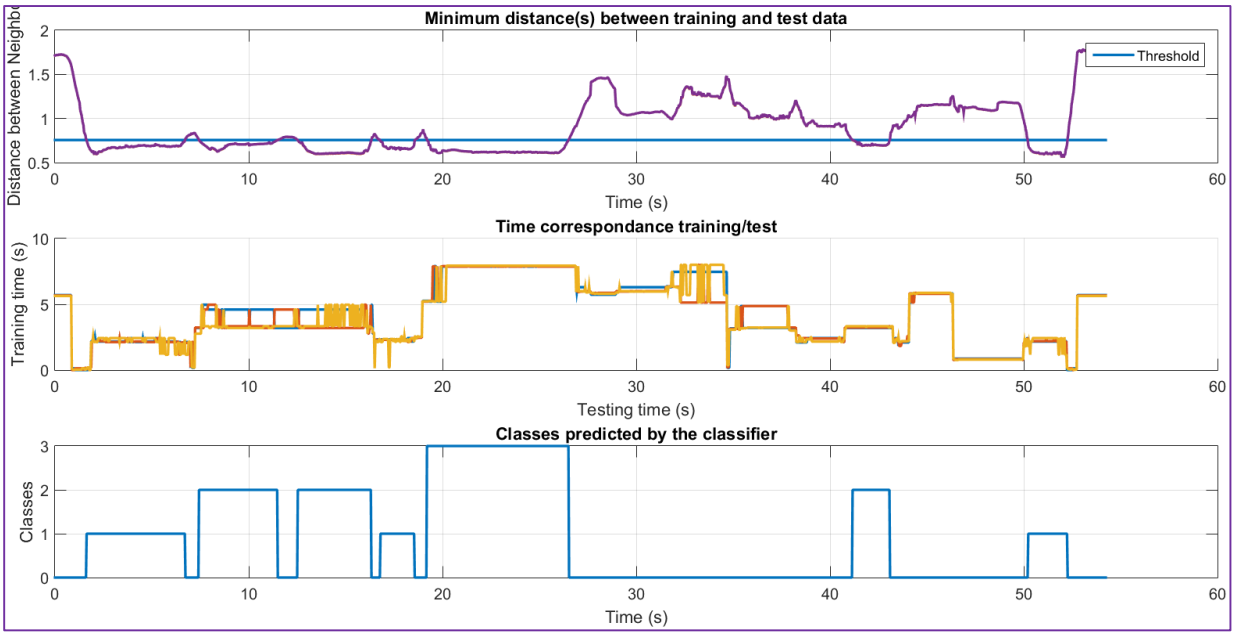

Figure 17:Premires résultats de la classification par méthode des K Nearest Neighbors adaptée

Le premier graphe est la représentation de la distance entre l'échantillon de test et son ou ses plus proche(s) voisin(s) dans les données d'apprentissage. Le seuil y est représenté afin d'avoir un témoin visuel pour le rejet des positions trop éloignées.

Le second graphe représente la correspondance temporelle entre l'échantillon de test et celui d'apprentissage. Par exemple, à 22 s du test, la position est proche de celle qui a été effectuée à 7 s dans l'apprentissage. Nous savons donc relier une position faite dans le test à une position de l'apprentissage.

Le dernier graphe est le plus intéressant puisqu'il s'agit du résultat de la classification. On peut voir que le rejet des positions éloignées fonctionne. Par exemple autour de 30 s on s'aperçoit que la distance augmente beaucoup (cela correspond aux mouvements aléatoires) et ces positions ne sont pas classées. Si l'on observe le graphe 2 on voit que l'on trouve tout de même un ou des plus proche(s) voisin(s), mais on le rejette car au-dessus du seuil.

On peut maintenant observer la répartition spatiale des classes afin de vérifier que nous avons bien rejeté les mauvaises positions et classifié les bonnes aux bons endroits. Le graphe résultant est disponible ci-après (*Figure 18*) :

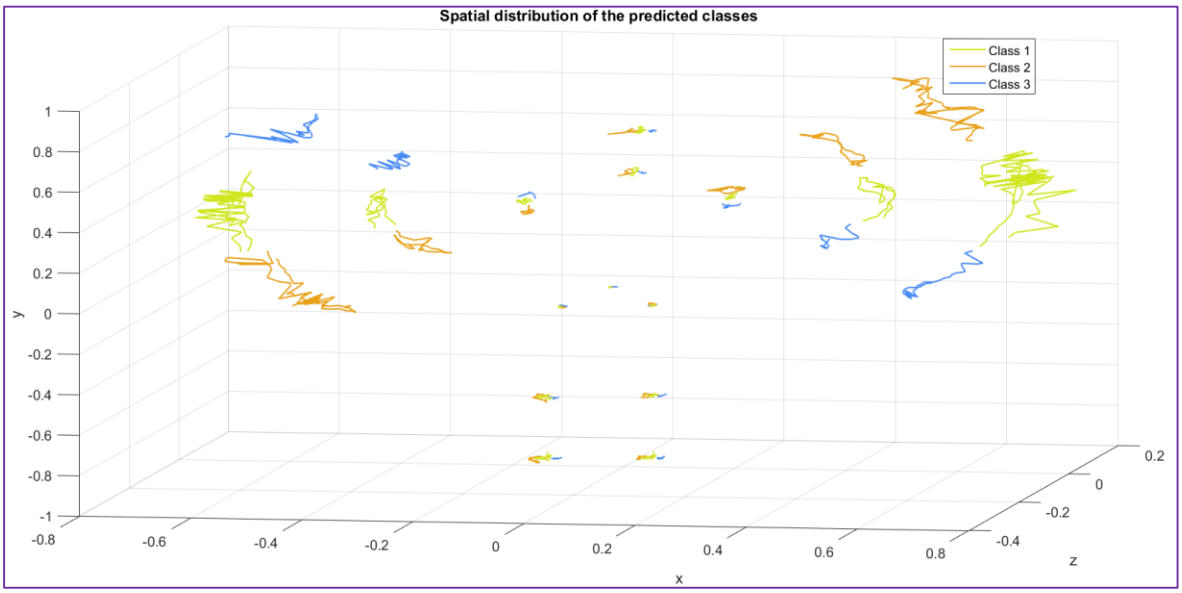

Figure 18:Représentation en trois dimensions des classes du mouvement

On observe bien nos trois classes, facilement reconnaissables. Elles correspondent bien aux positions de l'apprentissage. Cette visualisation simple permet d'avoir un retour direct sur la répartition des classes, facilitant ainsi le diagnostic.

On peut montrer, avec un cas plus complexe, que notre classificateur fonctionne même avec peu de points d'apprentissage par classe. Le test se compose de 10 classes qui dérivent toutes des positions utilisées pour l'essai précédent (certaines avec les jambes tendues, pliées, le corps incliné, etc.).

La figure page suivante (*Figure 20*) montre les résultats obtenus. Sur le premier graphe on remarque que les distances ne présentent pas autant de variations que précédemment. Cela s'explique par le fait que les 10 positions sont proches entre elles : en passant de l'une à l'autre on s'éloigne peu de la première pour passer à la deuxième. De ce fait le choix du seuil est ici délicat.

En regardant les résultats de la classification sur le dernier graphe, on peut constater que certaines positions n'ont pas été gardées très longtemps. À cause d'un apprentissage sommaire, les classes sont plus strictes et sont donc plus difficilement atteignables.

La représentation spatiale (*Figure 19*) qui suit permet d'ajouter une vérification à ces résultats. En effet, même s'il y a peu de points identifiés par classe, on peut tout de même se rendre compte que chaque position est bien définie, délimitée dans l'espace.

Figure 20: Résultats de la classification avec la méthode des K Nearest Neighbors modifiée sur un cas plus complexe

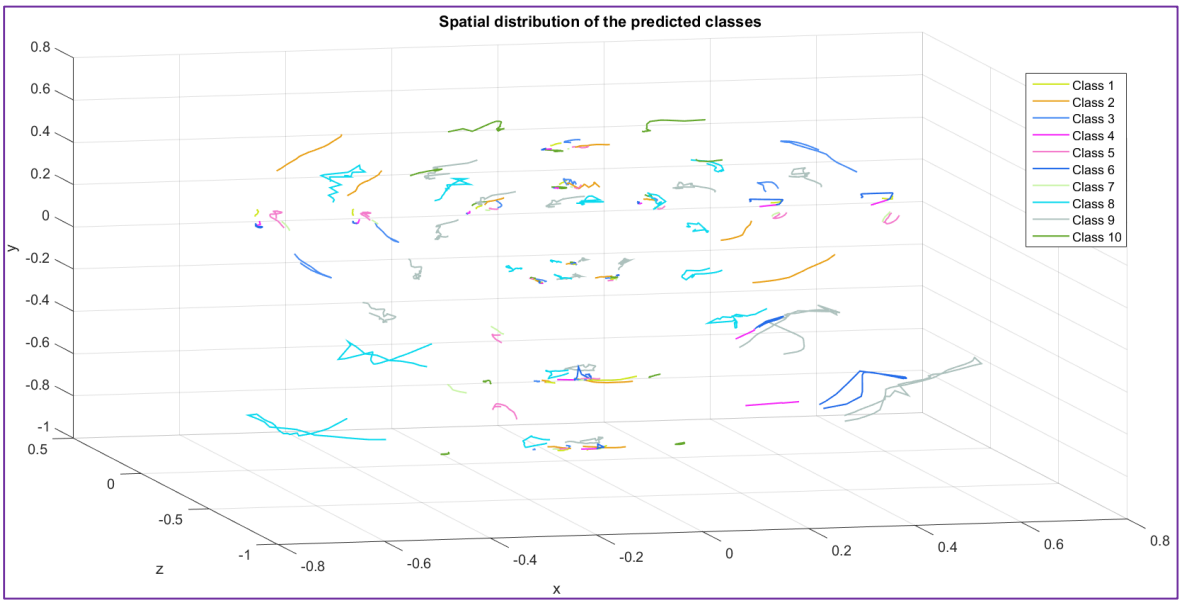

Figure 19: Représentation en trois dimensions des classes du cas plus complexe

V.4. D'autres outils sont disponibles

Jusqu'à présent, nous n'avons présenté que des implémentations sous Matlab des différents algorithmes, en utilisant des bibliothèques proposées ou en développant nos propres améliorations des méthodes disponibles. Il existe toutefois de nombreux autres logiciels et outils dans la communauté des développeurs et des personnes travaillant dans le domaine du *big data*. Ces solutions peuvent être gratuites et libres, ou bien sous licence, et parfois payantes. En solutions libres pour le *big data*, nous pouvons citer par exemple :

- KNIME[4], basé sur Eclipse et donc sur le langage Java, mais permettant l'utilisation d'une interface graphique pour le *data mining* et le *machine learning* ;
- Project R[5], qui est un projet GNU, écrit dans un langage de programmation propre (le langage R), basé sur un mélange de C et de Fortran, et qui est extrêmement courant dans le domaine des statistiques ;
- Orange[6], le seul que nous ayons eu l'occasion de tester, qui propose une interface graphique à base de widgets et de connexions entre ces widgets, ainsi que des bibliothèques pour travailler en ligne de code en Python, le tout reposant sur une sous-couche en C++.

Nous avons consacré une partie du projet à étudier cette solution alternative que constitue Orange. Toutefois, nous n'avons pas pu mener les expérimentations à leur terme. En effet, l'interface graphique d'Orange est relativement récente, surtout dans sa dernière version développeur (v3.2), et la documentation, les références et la communauté qui permettent traditionnellement de se familiariser avec un langage restent assez rares et pauvres. De plus, l'utilisation de la partie programmation a été freinée par le temps restant au moment de l'exploration de nouvelles solutions et de notre méconnaissance du langage Python.

Après plusieurs essais des deux possibilités (scripts Python ou interface graphique), nous nous sommes finalement limités à l'interface graphique, tout en ayant une connaissance globale du fonctionnement interne des classes Python du programme. L'intérêt que nous avons trouvé quant à l'utilisation de l'interface graphique de ce logiciel repose essentiellement pour ses qualités d'affichage des résultats lors des phases d'apprentissage (supervisé ou non). Par contre, bien que nous ayons pu commencer à expérimenter le logiciel sur les méthodes de prédiction, nous n'avons pu aboutir à des résultats certains, fiables et convaincants.

Tout d'abord, notons que l'on ne peut pas directement utiliser les fichiers « .txt » générés lors de la capture par l'interface C# présentée précédemment. Le logiciel Orange travaille plutôt avec les formats classiques de stockage de données : .tab, .csv, .xls… Nous aurions bien sûr pu ouvrir les fichiers directement en Python et effectuer les transformations de format nécessaires dans un script, mais nous avons effectués ces changements manuellement pour effectuer nos essais, bien qu'il soit possible de l'automatiser.

[4] www.knime.org
[5] www.r-project.org et cran.irsn.fr
[6] orange.biolab.si

La figure suivante (*Figure 21*) présente le schéma général utilisé pour effectuer nos tests sous Orange.

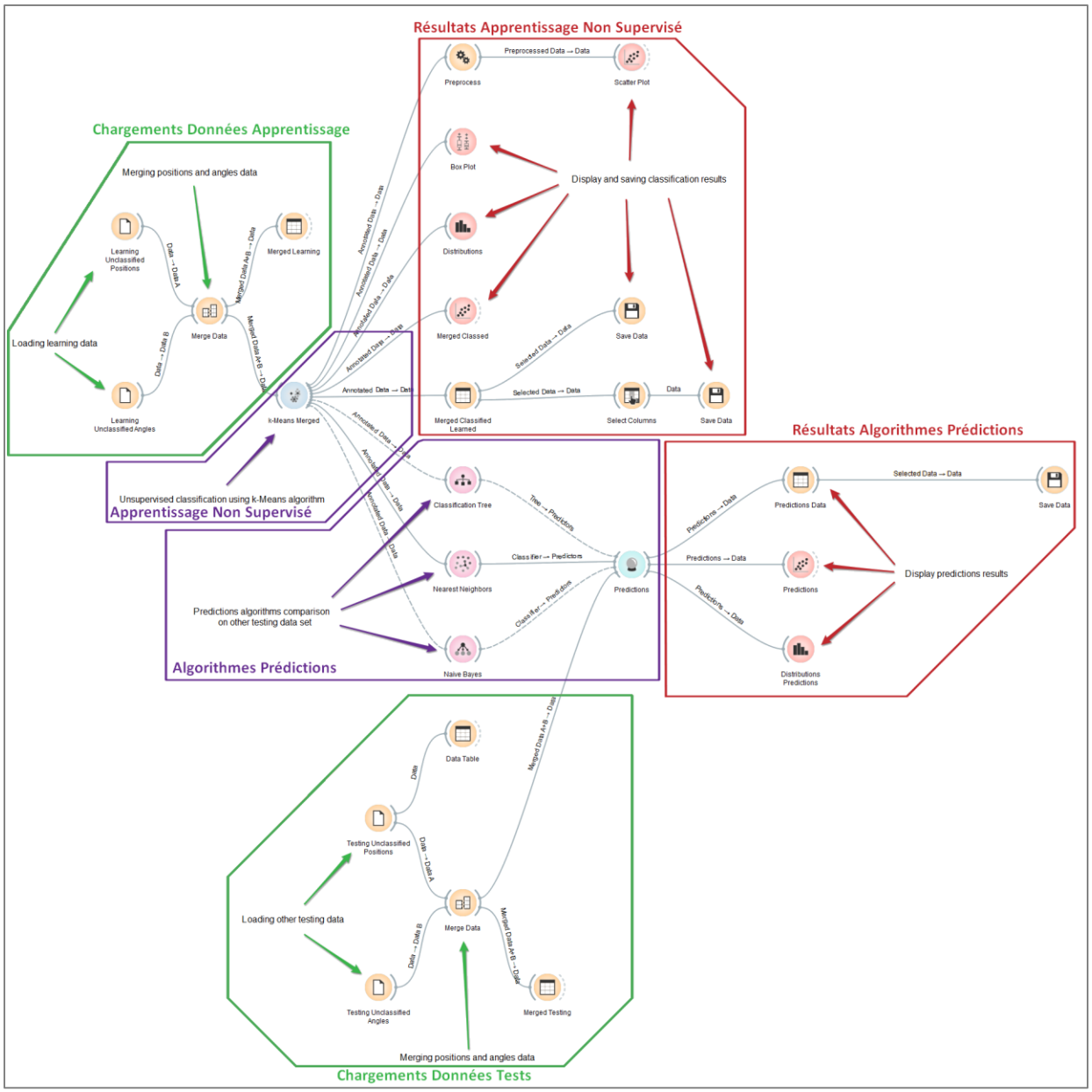

Figure 21:Schéma de principe général de l'utilisation du logiciel Orange

On retrouve d'abord, en vert, deux zones comportant des widgets qui permettent de charger et de mettre en forme les données nécessaires, d'une part pour l'apprentissage, et d'autre part pour la prédiction. Il faut noter que l'on doit assembler les données de positions et d'angles si on veut traiter les deux simultanément, afin qu'elles soient considérées comme faisant partie d'un même jeu de données.

On a ensuite un widget (en violet) correspondant à un apprentissage non supervisé. On utilise ici l'algorithme dit de K-Means vu précédemment en implémentation sous Matlab. Pour rappel, il s'agit d'un algorithme non supervisé mais qui nécessite quand même de préciser le nombre de *clusters* que l'on recherche. Toutefois, le widget proposé par Orange est plus souple dans la mesure où on peut spécifier un intervalle de nombre de *clusters* possible. On obtient alors un « score » de correspondance pour chaque

nombre de *clusters* possible, et on peut visualiser chaque résolution, le widget proposant la meilleure correspondance par défaut. Notons que l'on choisit l'algorithme de K-Means car nous avons déjà pu l'étudier auparavant et qu'il est particulièrement bien adapté à de grands jeux de données.

Le bloc adjacent (en rouge) propose un ensemble de plusieurs widgets permettant l'affichage ou la sauvegarde des données. On retrouve en particulier un affichage classique de répartitions des points en deux dimensions (pour deux variables donc), un affichage de la distribution des valeurs de chacune des variables (regroupement possible par *cluster*), un affichage des valeurs moyennes, médianes et variances des variables… Des exemples de résultats seront donnés dans un prochain paragraphe.

Le second bloc violet correspond au test de plusieurs algorithmes de prédiction, comme le *Nearest Neighbors* vu précédemment ou l'algorithme de Bayes qui utilise des probabilités d'appartenir à une classe plutôt que des évaluations de distances comme dans le *Nearest Neighbors*. Pour cela, les widgets des différents algorithmes reçoivent les données classifiées par l'étage d'apprentissage, et le widget de prédiction reçoit également les données des instances de tests. On récupère en sortie les attributions des nouvelles données aux classes initiales par les différentes méthodes de prédiction. C'est à ce moment que notre étude a été limitée, dans la mesure où nous ne maîtrisions pas suffisamment le logiciel pour, d'une part, contrôler les paramètres des différents algorithmes (pour par exemple pouvoir rejeter des points trop éloignés des classes initiales comme vu précédemment pour l'implémentation personnalisée du *K-Nearest Neighbors*), ou d'autre part pour modifier les caractéristiques des données de sortie du widget de prédiction (il aurait donc nécessité un traitement externe que nous n'avons pas eu l'occasion de mettre en place). Le second bloc rouge permet tout de même d'afficher les résultats bruts obtenus en prédiction, bien que nous ne soyons pas parvenus à maîtriser parfaitement le comportement des widgets utilisés en prédiction.

Pour exemple, nous reprendrons l'exemple précédemment étudié des trois positions classifiées sous Matlab (bras tendus à l'horizontale, bras gauche levé et droit baissé, bras gauche baissé et droit levé), car on connaît assez précisément les résultats souhaités afin de réaliser des comparaisons des résultats obtenus.

L'objectif est de vérifier que l'algorithme d'apprentissage trouve une correspondance pour trois *clusters*. La figure ci-dessous (*Figure 22*) montre bien que l'on a laissé un large intervalle à l'algorithme

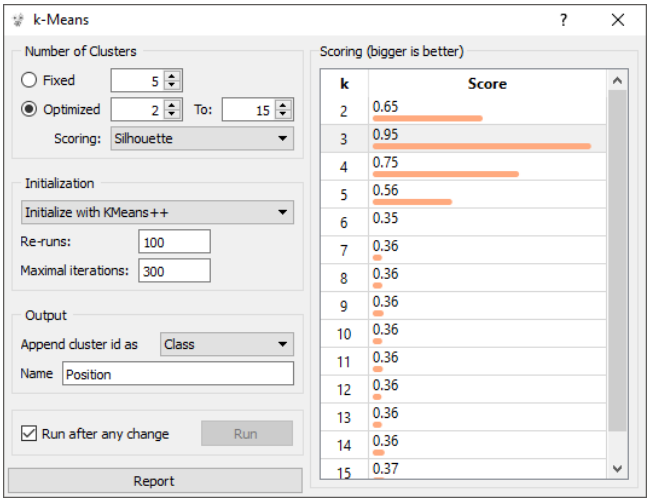

Figure 22:Résultats des apprentissages par l'algorithme de K-Means du logiciel Orange pour plusieurs nombre de clusters

(entre deux et quinze *clusters*), et que le score est bien nettement supérieur pour trois *clusters* (un score de 1 signifiant une correspondance parfaite).

Les figures suivantes (*Figure 24* et *Figure 23*) montrent un extrait des possibilités d'affichage de résultats du logiciel.

Tout d'abord, on peut afficher le graphe en deux dimensions de n'importe quelle variable en fonction d'une autre, colorer les points en fonction de leur appartenance à une classe, ou en fonction du temps (le temps n'intervient pas dans la classification, mais on le stocke quand même comme une « méta » donnée, c'est-à-dire une donnée annexe, non utile à la résolution mais qui peut quand même apporter une information sur les instances). Par exemple, dans le cas des positions étudiées, on peut afficher l'évolution de la hauteur (projection suivant l'axe y donc) du point du squelette correspondant à un coude, le droit par exemple, en fonction du temps, ou encore en fonction de la projection suivant l'axe X. Notons que dans Orange, nous n'utilisons que les données de position relative, et les angles calculés au moment de la capture, mais pas les processus de normalisation vus dans l'implémentation Matlab de l'algorithme du plus proche voisin.

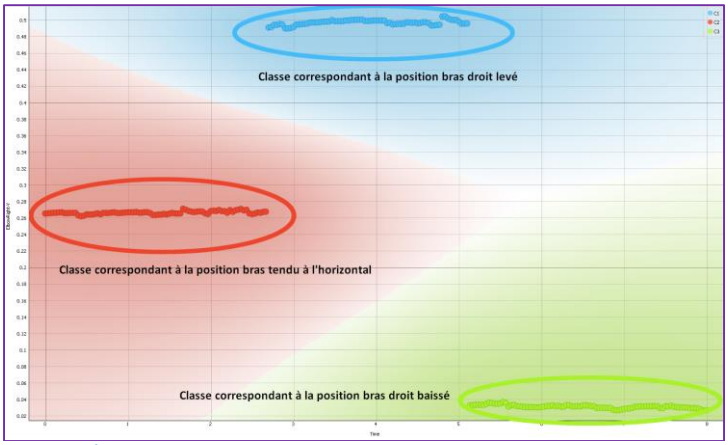

Figure 24:Évolution de la composante en Y du coude droit au cours du temps

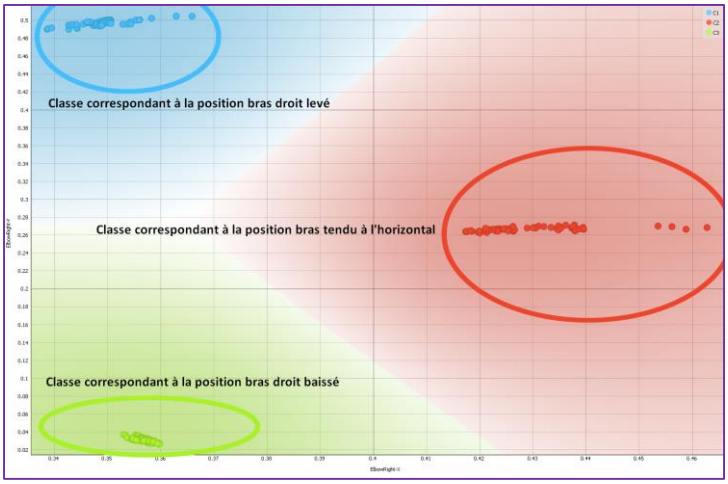

Figure 23:Évolution de la composante en Y du coude droit en fonction de la composante en X

On constate que la méthode a bien isolé trois groupes d'instances, et qu'elles correspondent effectivement aux trois positions successives décrites précédemment. L'affichage d'autres grandeurs (autres positions ou angles) confirme aussi la validité ces observations. Le logiciel permet donc de bien visualiser les zones de l'espace des variables qui correspondent à chaque *cluster*, et qu'elles sont bien séparées. Cette séparation se retrouve sur les graphes de distribution des variables pertinentes. Si l'on reprend le coude droit, on obtient la répartition suivante (*Figure 25*).

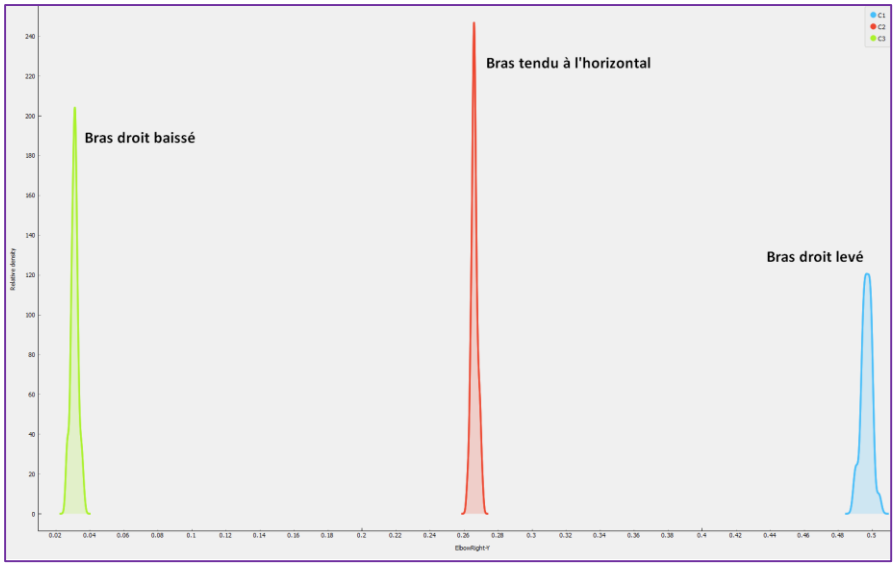

Figure 25:Répartition des valeurs de la composante en Y du coude droit

Enfin, on peut consolider ces observations avec un autre graphe qui présente lui aussi la répartition des valeurs, mais sous une autre forme, comme le montre la figure suivante (*Figure 26*).

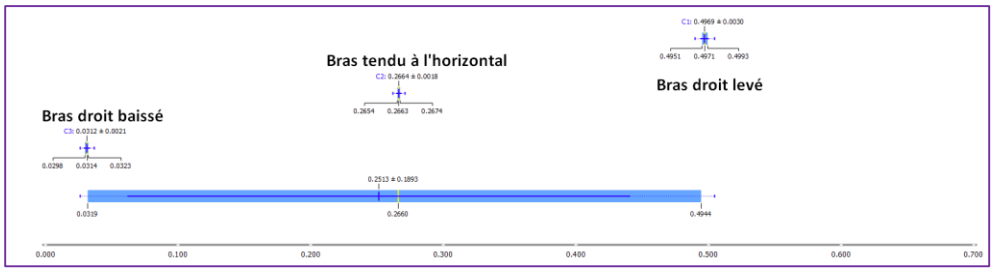

Figure 26:Répartition des valeurs de la composante en Y du coude droit (toutes les valeurs, et par cluster)

On peut ici visualiser les valeurs moyennes, médianes, les variances des valeurs par classe, et pour l'ensemble des points de la variable. Ces informations sont complémentaires et un peu redondantes avec les précédentes, mais cela permet de montrer qu'Orange est un outil relativement puissant en termes de visualisation de résultat, ce qui constitue un avantage considérable pour effectuer l'analyse a posteriori des résultats, ainsi que pour en rendre compte à des personnes tierces, pas forcément spécialistes du monde des méthodes d'apprentissage ou même du diagnostic.

Une fois l'apprentissage effectué, nous aurions souhaité pouvoir tester et comparer les algorithmes de prévisions. Nous n'avons pu mettre véritablement en place que la méthode du plus proche voisin. La figure suivante (*Figure 27*) montre des résultats pour le même jeu de données de test que précédemment, c'est-à-dire avec les bonnes positions, entrecoupées de mouvements aléatoires. Ici, nous ne pouvons appliquer que la méthode des *K Nearest Neighbors* brute, et pas éliminer les points trop éloignés des classes possibles. On obtient tout de même des résultats globalement cohérents, mais il n'y a pas de rejet des points aberrants, il faudrait effectuer un post traitement des données, une expertise. On peut tout de même observer les résultats obtenus pour l'évolution de la coordonnée Y en fonction de la coordonnée de X, toujours du coude droit au cours de la capture.

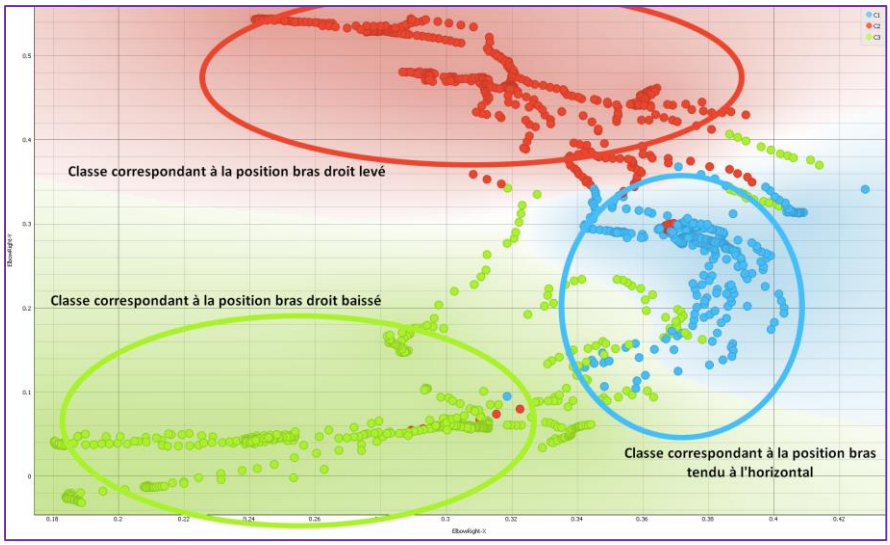

Figure 27:Prédiction de la correspondance du mouvement de test avec les clusters d'apprentissage

Notons par contre, que si des points semblent aberrants à première vue dans une telle représentation (point d'une classe au milieu d'un nuage d'une autre classe), il faut garder à l'esprit qu'il s'agit d'une représentation plus que partielle des instances, car il y a bien entendu beaucoup plus que deux variables (les positions en X, Y et Z de chaque point caractéristique du squelette et les angles des articulations associées).

Ainsi, compte tenu du temps imparti pour notre apport au projet pendant ce stage, nous avons dû interrompre nos analyses à ce niveau concernant le logiciel Orange. Cependant, cela nous a permis de voir qu'il existait toujours plusieurs solutions pour appréhender la résolution d'un problème donné, qui ont toutes leurs atouts et leurs inconvénients (simplicité relative d'utilisation, documentation disponible, performances et vitesse des traitements, nécessité de pré ou post traitement…) et qu'il faut être à même d'explorer et de choisir une solution adaptée qui permette une analyse pertinente des résultats de classification obtenus.

VI. Conclusion

L'objectif initial de ce stage, à savoir l'analyse des mouvements de sportifs pour prévenir les mauvaises pratiques, a évolué et n'a pas été entièrement rempli. Néanmoins, du chemin a été parcouru dans la compréhension tant du fonctionnement du dispositif Kinect, que dans le monde du diagnostic appliqué à cet objectif. Ces clés ont été développées dans le but de pouvoir être réutilisées dans la continuité du projet. Les contraintes de temps et d'équipe n'ont pas permis d'amener à l'aboutissement de toutes les pistes explorées, mais elles constituent indéniablement un premier état de l'art nécessaire dans un tel projet.

A l'issue de cette période de stage, nous sommes capables, au travers des outils développés, de détecter des positions faites par un sujet et de les comparer à une base de données afin de dire si elles sont correctes ou non. Grâce au traitement effectué il est possible d'ajouter à ces données la notion de vitesse pour tenter de caractériser des mouvements.

La méthode la plus aboutie que nous ayons mise en place (KNN) est entièrement supervisée. Au travers de nos investigations sur les autres moyens de classification nous avons tout de même exploré la possibilité de faire l'apprentissage des classes à l'aide d'une méthode non supervisée, puis, à partir de ces données, utiliser notre classificateur supervisé, rendant le tout, non supervisé.

Enfin en ce qui concerne l'aspect matériel, nous avons rencontré un certain nombre de limitations, qui se sont avérées relativement bloquantes au vu de l'objectif global du projet. Dans la limite du possible nous pensons que ce dernier pourrait tirer parti d'un passage à Kinect v2 (pour Xbox One). La fréquence d'images étant plus élevée il serait ainsi possible d'observer plus finement les mouvements. La définition augmentant et la technologie avançant, le suivi des articulations serait bien plus précis. Aussi, l'angle de vue plus étendu permettrait de meilleures conditions de tests, dans la mesure où le sujet serait plus libre de faire ses mouvements.

Dans l'ensemble ce sujet a été pour nous l'occasion de découvrir un nouveau domaine dont nous n'avions pas connaissance, à savoir le diagnostic, ce qui pourra s'avérer utile dans notre vie future d'ingénieur, car, comme nous l'avons vu, le traitement de grands jeux de données est devenu omniprésent et permet de déduire beaucoup d'informations. De plus nous avons pu approfondir bon nombre des connaissances acquises en école comme la programmation orientée objet, l'usage de MATLAB, le travail en équipe, la gestion de projet... Tous ces savoirs précieux seront certainement valorisables très prochainement en entreprise.

Nous tenons enfin à remercier nos responsables Euriell Le Corronc et Elodie Chanthery pour leur accueil au sein de l'équipe DISCO et pour leur accompagnement au cours de ce projet.

Table des figures