

RAPPORT DE STAGE MASTER RVSI

Interactions Homme-NAO à travers des gestes emprunts d'émotions

STAGIAIRE : MELLE ALICE JOURLIN, ENCADRANT : MR MALIK MALLEM



Du 26 Février au 24 Août 2018

Table des matières

1 Résumé	4
1.1 Le projet	4
1.2 Réalisations du stage	5
2 L'entreprise d'accueil	6
2.1 Laboratoire IBISC	6
2.2 Cadre de travail	6
2.3 Outils	7
3 Problématique	11
4 Etat de l'art	12
4.1 Introduction	12
4.2 Applications	13
4.2.1 Vidéo surveillance	13
4.2.2 Interaction homme-machine	13
4.2.3 Domaine médical	13
4.2.4 Récupération vidéo	14
4.3 Le geste	14
4.4 Détection et suivi	14
4.5 Extraction des données	19
4.6 Reconnaissance de gestes	20
4.6.1 Machine Learning	21
4.6.2 Clustering	21
5 Réalisation de la base de données	24
5.1 Utilisation de XSens et MVN Analyze	24
5.2 Acquisition de données	27
5.3 Analyse des résultats	27
5.3.1 Analyse mathématique	27
5.3.2 Etude statistique	28
6 Méthode de reconnaissance de gestes	30
6.1 Apprentissage	30
6.1.1 Acquisition des données	30
6.1.2 Extraction de caractéristiques	30

6.1.3	Echantillonnage	31
6.1.4	Quantification	31
6.2	Reconnaissance	31
6.2.1	Traitement des données	31
6.2.2	Comparaison	32
6.3	Résultats	32
7	Liaison XSens-NAO	33
7.1	Situation initiale	33
7.2	Réseau	33
7.3	Lecture de fichier .csv	34
7.4	Comparaison XSens - <i>Kinect</i>	34
7.4.1	Comparaison à l'usage	34
7.4.2	Protocole avec Unity	35
8	Conclusion et perspectives	37
Bibliographie		38
Glossaire		41
Annexes		42
Extrait du manuel d'utilisation	42	
Manuel d'utilisation du réseau avec MVN Analyze	51	
Client et serveur UDP pour linux	67	
Lecture de fichier .csv	70	

Remerciements

Je tiens à remercier :

Monsieur Malik MALLEM pour avoir été mon maître de stage et encadrant pendant ces 6 mois

Insaf AJILI et Zahra RAMEZANPANAH, doctorantes avec lesquelles j'ai collaboré

L'IBISC pour m'avoir fourni du matériel sans lequel je n'aurais pas pu mener mes recherches, ainsi que de bonnes conditions de travail

L'ENSIIE et l'Université d'Evry Val d'Essonne pour l'opportunité de faire ce stage



1 Résumé

1.1 Le projet

Le projet auquel j'ai participé est composé de deux grandes parties. La première consiste à contrôler un robot NAO via différents gestes reconnus grâce à des capteurs de mouvements (d'une part RGB-D avec une caméra *kinect*, et d'autre part inertIELS avec XSens). La seconde est la reconnaissance d'émotions à travers des gestes, que le robot NAO saura interpréter. Ainsi, le robot réagira différemment au même geste s'il est réalisé avec colère ou avec tristesse.

Contrôle de NAO par la *kinect*

Ce premier mode d'utilisation permet au robot NAO de reconnaître les actions des utilisateurs et d'agir en conséquence.

Tout d'abord, il y a une acquisition de données via la *kinect*. Les mouvements de l'utilisateur sont ainsi détectés. Pour les interpréter en temps réel, la *kinect* est intégrée directement au logiciel de traitement des données, créé par un ancien stagiaire. Ce logiciel analyse les données en temps réel et utilise le système de contrôle linux-NAO ROS (Robot Operating System) pour transmettre les ordres au robot NAO. Différents modes de traitements sont implémentés, et il est possible de passer de l'un à l'autre grâce à des mouvements de jambe réalisés par l'utilisateur. Ces modes permettent de faire aussi bien des mouvements par mimétisme (le robot reproduit fidèlement les mouvements de l'utilisateur) que de la télé-opération (le robot reconnaît les mouvements de l'utilisateur et agit en conséquence).

L'objectif est alors de remplacer l'acquisition des données, initialement faite par la *kinect*, par une acquisition via XSens, puis de comparer les avantages et les inconvénients de ces deux technologies.

Reconnaissance de gestes expressifs

Une autre partie du projet était d'aider deux doctorantes dans leur travail sur la reconnaissance de gestes, et notamment de gestes expressifs.

La première étape a été de créer une base de données pour pouvoir réaliser la classification, mais surtout pour pouvoir réaliser une étude statistique. Cette base de données se compose de vidéos enregistrées par le logiciel MVN Analyze qui est associé au système XSens. L'étude statistique qui lui est associée a pour but d'analyser la justesse des gestes. En effet, les gestes utilisés pour contrôler le robot NAO vont être ici réalisés quatre fois chacun, avec quatre émotions différentes : colère, calme, joie et tristesse. Il faut donc vérifier que les attitudes que nous supposons représenter ces émotions sont interprétés de la même manière par les futurs utilisateurs.

Une fois la liste des gestes établie, des algorithmes et méthodes ont été utilisés pour réaliser une classification. Ceci est détaillé dans la partie 6, intitulée Méthode de reconnaissance de gestes.



NAO et caméra *kinect*

1.2 Réalisations du stage

Mon stage a pour objectifs de manipuler nombre de technologies et de faire avancer le projet d'interaction homme-NAO. Pour cela, ma première tâche a été de me familiariser avec le projet et de prendre connaissance de ce qui a déjà été fait. Ensuite, j'ai dû me familiariser avec XSens, équipement acquis pour la première fois par le laboratoire IBISC, pour pouvoir utiliser cette technologie lors de la reconnaissance de gestes expressifs. J'ai ensuite eu pour tâche de reprendre le travail de télé-opération de NAO par reconnaissance du geste pour pouvoir faire l'acquisition des données non plus par la caméra *Kinect*, mais par XSens. Ceci a posé de nombreux problèmes, qui n'ont pas pu être résolus durant le stage. Il est actuellement impossible de téléopérer NAO en temps réel. Mais j'ai réalisé un programme qui permet de lire les données enregistrées par XSens, ce qui permettra de téléopérer NAO en différé.

2 L'entreprise d'accueil

2.1 Laboratoire IBISC



Le laboratoire IBISC (Informatique, Bio Informatique et Systèmes Complexes) qui m'a accueillie est organisé en 4 équipes de recherche et se compose de plus de cinquante enseignants chercheurs et de plus d'une cinquantaine de doctorants. Les recherches conduites au sein d'IBISC ont pour perspective de développer des méthodes, formalismes et réalisations pour la compréhension des systèmes complexes, vivants ou artificiels.

Son projet scientifique s'articule autour de deux axes fédérateurs :

- L'axe STIC et Vivant a pour objet le développement à la fois de la biologie computationnelle et de la bio-informatique ainsi que de l'assistance à la personne.
- L'axe STIC et Smart System a pour objet l'étude et la conception de systèmes autonomes et intelligents dans le cadre d'environnements ouverts.

C'est sur ce second axe que se situe mon stage.

Associant recherches pluridisciplinaires, fondamentales et appliquées, et ancré en Sciences et Technologie de l'Information et de la Communication (STIC), le laboratoire IBISC se positionne donc comme faisant partie du département STIC de l'université Paris Saclay.

2.2 Cadre de travail

J'ai travaillé au sein d'un groupe composé de deux doctorantes, Insaf AJILI et Zahra RAMEZANPANAH, sous la responsabilité de monsieur MALLEM. Nous travaillions toutes les trois sur le même sujet, avec une salle dédiée contenant tout le matériel nécessaire.

Etant au sein de l'IBSIC, j'ai eu la possibilité d'accéder à beaucoup de matériel peu ou pas accessible au grand public, que ce soit pour travailler dessus ou tout simplement le découvrir.

2.3 Outils

NAO

NAO est un robot humanoïde autonome, programmable et mesurant environ 58 cm, développé par la société française Aldebaran Robotics, qui est actuellement une filiale de SOFTBANK. Il a été présenté pour la première fois au public fin 2006 et est beaucoup utilisé aujourd’hui dans les laboratoires de recherche et les universités à travers le monde.

Caractéristiques techniques	
Hauteur	58 cm
Poids	4,8 kg
Autonomie	90 min
Degrés de liberté	14 à 25
Processeur	Intel ATOM 1,6 GHz (V4) ou AMD Geode 550Mhz (V3.3 ...)
Système d'exploitation intégré	Linux
Systèmes d'exploitation compatibles	Windows, Mac OS, Linux
Langages de programmation	C++ , Python , Java , MATLAB , Urbi , C , .Net
Connectivité	Ethernet, Wi-Fi (b,g,n)
Vision	2 caméras 920p, 30ips
Audio	4 Microphones

Caractéristiques

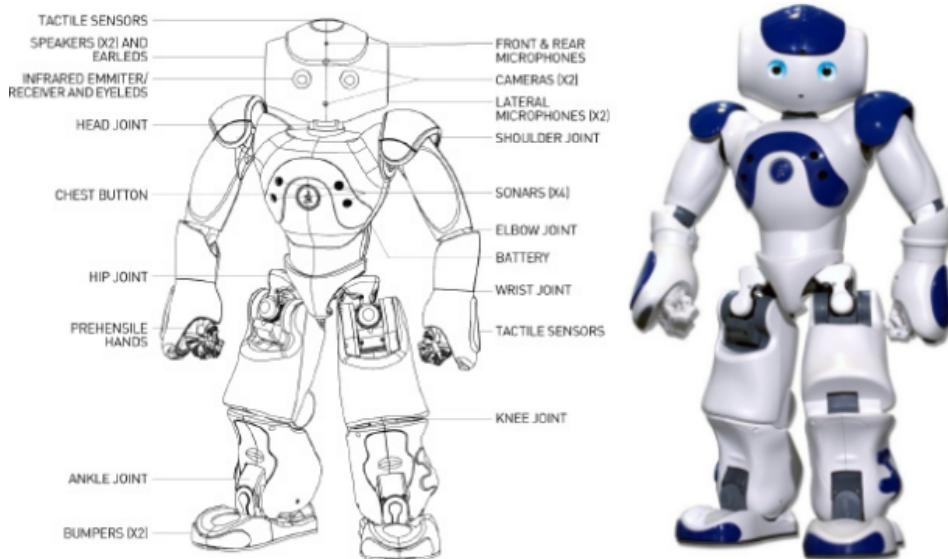
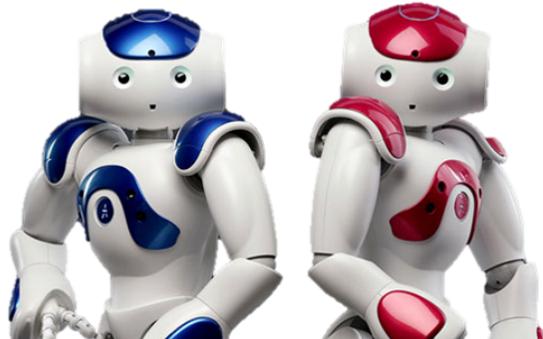


Schéma du robot NAO

Le robot NAO dispose d'une batterie mais du fait de sa faible autonomie, il est recommandé de laisser le robot branché sur secteur lorsque l'on travaille avec lui. Par ailleurs, NAO a besoin de se connecter au réseau. Il est possible de le connecter

en wifi, mais dans notre cas, il a été décidé de le connecter en filaire directement sur le PC linux.



Robots NAO

ROS

ROS (Robot Operating System) fournit des bibliothèques et des outils pour aider les développeurs de logiciels à créer des applications robotiques. Il fournit une abstraction matérielle, des pilotes de périphériques, les bibliothèques, les visualiseurs, le passage de messages, la gestion des paquets, et plus encore. ROS est sous licence open source, licence BSD. Il est donc utilisé sous linux, dans notre cas, pour utiliser le robot NAO.

Kinect

La caméra *kinect* est un périphérique initialement créé par Microsoft pour la Xbox 360, qui est utilisable également sur PC, qui est sorti en 2010.

Caractéristiques :

Capteur :

Lentilles détectant la couleur et la profondeur

Micro à reconnaissance vocale

Capteur motorisé pour suivre les déplacements

Champ de vision :

Champ de vision horizontal : 57 degrés

Champ de vision vertical : 43 degrés

Marge de déplacement du capteur : ± 27 degrés

Portée du capteur : 1,2 m – 3,5 m

Flux de données :

320 \times 240 en couleur 16 bits à 30 images par seconde

640 × 480 en couleur 32 bits à 30 images par seconde

Audio 16 bits à 16 kHz

Système de reconnaissance physique :

Jusqu'à 6 personnes

20 articulations par squelette

Audio :

Suppression de l'écho

Reconnaissance vocale multilingue

XSens et MVN Analyze

Xsens Technologies B.V. (ou Xsens) est un système de capture de mouvement en 3D. Il est basé sur un ensemble de microsystème électromécanique utilisant une technologie de centrales inertielles. Dans notre cas, nous utilisons la combinaison XSens Awinda. Ainsi, nous disposons de 17 centrales à placer sur le corps de l'utilisateur.



XSens Awinda en application

Pour pouvoir réaliser un suivi ou une acquisition des données, nous disposons du logiciel MVN Analyze. Ce logiciel n'est disponible que sous Windows mais est indispensable pour pouvoir utiliser XSens Awinda, puisque son utilisation commence par la détection des différents capteurs par le logiciel, puis une phase de calibration. Il est ensuite possible de faire de l'acquisition de données. Toutes ces étapes seront détaillées et expliquées par la suite.

3 Problématique

Dans notre monde contemporain, nous sommes entourés de robots. Que ce soit dans notre cuisine, dans les usines, dans les hôpitaux ou encore parmi les jouets pour enfants, il y en a partout. Ils prennent diverses formes, mixeur, machine d'assemblage, humanoïde ou encore animal, et leur intelligence se développe sans cesse.

Le type de robot auquel nous allons nous intéresser est le robot humanoïde. Il a comme caractéristique d'avoir un aspect le plus proche possible de l'être humain. Cela se caractérise par un corps le plus proche possible, notamment avec une tête, deux bras, deux jambes, des mains, des pieds... Il peut avoir moins de doigts, et a en général moins d'articulations que l'être humain. En revanche, certaines de leurs articulations peuvent permettre des mouvements inaccessibles à l'être humain. Certains ont pour but de ressembler le plus possible à l'être humain, tandis que d'autres cherchent simplement à évoquer l'humain.

Pour commander ces robots, comme les humanoïdes, on peut utiliser un système de commande à distance : la télé-opération. Cela permet de commander le robot qui peut se trouver loin du contrôleur. De nombreux domaines d'application existent car cela permet notamment de contrôler un robot qui se trouverait dans une zone dangereuse, comme les robots de déminage, ou des robots qui vont dans des endroits que l'homme ne peut tout simplement pas atteindre.

Enfin, si les robots ressemblent de plus en plus à des humains, il reste tout de même de nombreuses différences. L'un d'eux est l'absence de sentiments des robots. Non seulement ils n'en éprouvent pas, mais ils ne sont actuellement pas non plus capables de les identifier, ni à travers la voix de la personne qui est en face d'eux, ni à travers les gestes de cette personne.

Mon projet concerne la télé-opération d'un robot qui serait capable de détecter les émotions de la personne qui le téléopère. Un choix avait déjà été fait quant à la méthode de télé-opération développée dans notre cas, mais il me faudra tout de même m'intéresser aux différentes possibilités, afin de comprendre les choix qui ont été fait. C'est d'autant plus important que je devrais m'interroger sur la technologie la plus performante pour notre projet entre *kinect* et XSens et que cela permet donc de mettre en lumière différents critères à prendre en compte. La seconde interrogation concerne les émotions en elles-mêmes. Lesquelles choisir, sur quels gestes ? Comment les exprimer pour qu'à la fois le robot et les êtres humains les reconnaissent ? Quels tests mener pour vérifier les hypothèses de reconnaissance ?

4 Etat de l'art

4.1 Introduction

Dans notre monde emprunt des nouvelles technologies, et compte tenu de l'essor de la vision par ordinateur et de la robotique, l'existence de l'interaction avec les robots est obligatoire et vitale. De fait, des progrès significatifs ont été faits en matière de machine learning et d'interaction homme-machine.

L'idée principale est de permettre aux ordinateurs la compréhension des ordres provenant des humains. Pour atteindre ce but, il faut apprendre à l'ordinateur à comprendre notre langue, à reconnaître les émotions, les expressions faciales et les gestes. Les actions humaines, à cause de leur complexité, peuvent être réparties en quatre catégories [3] : gestes, actions, interactions et activités de groupe, qui sont définies comme suit :

- Geste : mouvement rudimentaire d'une partie du corps qui sont des composants atomiques, comme de secouer une jambe ou frapper avec une main.
- Action : les activités d'une personne qui peuvent être composées de plusieurs gestes qui sont organisés temporellement, comme la course ou un saut.
- Interaction : actions bilatérales entre deux personnes/objets ou plus, comme se battre ou se serrer la main
- Activité de groupe : activités qui sont réalisées par un groupe de plusieurs personnes ou objets, comme le football ou tout autre sport d'équipe.

Il est également nécessaire de mentionner la publication réalisée suite à la première partie du projet que j'ai rejoint. [8] expliquent la démarche du projet et donnent de nombreuses pistes pour comprendre les choix qu'ils ont faits. Le point de départ de cette publication et donc du projet est le constat que même si c'est un travail commencé il y a longtemps, il est encore difficile de faire suivre les mouvements d'un humain à un robot humanoïde, à cause d'une part de la programmation qui n'est pas évidente, et d'autre part l'équilibre du robot, qui est très différent de celui du corps humain. Pour essayer de résoudre ces problèmes, il est possible de commencer par enregistrer les mouvements humains à l'aide de matériel de motion capture, puis de traiter mathématiquement les données ainsi acquises avant de les transmettre au robot. Les solutions de motion capture sont déjà nombreuses.

Différents systèmes de motion capture ont déjà été utilisés dans le cadre de travaux de téléopération, notamment pour téléopérer des robots humanoïdes comme le robot NAO. Par exemple, [11] ont utilisé XSens pour pouvoir enregistrer les mouvements humains à transmettre à leur robot NAO. Grâce à une analyse des données fournies par XSens en temps réel et à un calcul pour adapter ses valeurs aux articulations et au centre de gravité du robot, le robot prend exactement les mêmes positions que son téléopérateur.



NAO imite un mouvement complexe de son téléopérateur

4.2 Applications

4.2.1 Vidéo surveillance

De nos jours, les caméras de surveillance font partie de nos vies et sont utilisées partout : dans les places publiques comme les magasins, les aéroports, les stations de train, dans les rues pour obtenir le trafic routier, mais également dans certaines voitures intelligentes [2] pour assurer la sécurité, ce qui nécessite d'analyser les comportements humains et par conséquence avoir une reconnaissance des actions humaines automatique.

4.2.2 Interaction homme-machine

L'interaction homme-machine fait référence à l'interaction entre les hommes et les technologies informatiques qui sont présentes dans la vie quotidienne de chacun, comme les industries. Le capteur Microsoft Kinect est un dispositif qui entre dans cette catégorie, et il permet à l'utilisateur d'interagir avec lui à travers des gestes ou des commandes vocales. L'exploration de la relation homme-machine est rapidement devenu l'un des domaines de recherche les plus importants.

4.2.3 Domaine médical

Compte tenu de l'augmentation de l'âge de la population dans le monde, la reconnaissance automatique des actions humaines et les analyses comportementales sont de plus en plus importants pour pouvoir suivre et surveiller les personnes âgées. Surveiller les changements de leurs habitudes comme le déplacement, la nutrition ou leur sommeil permet aux médecins de trouver de nouvelles stratégies pour mieux les soigner.

4.2.4 Récupération vidéo

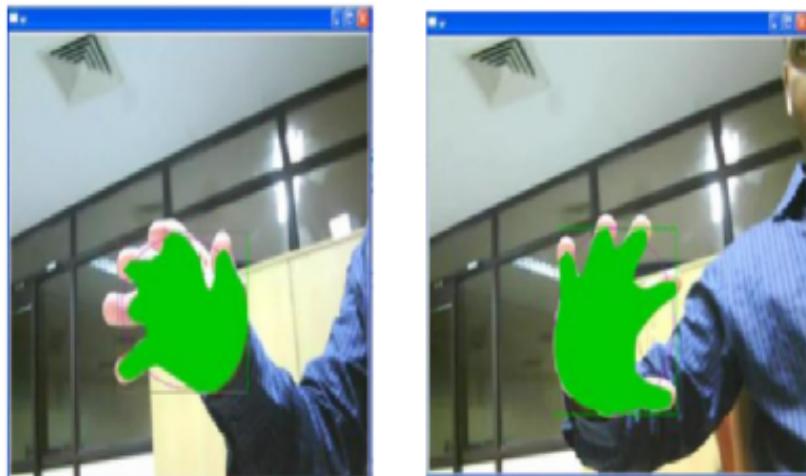
De nos jours, à cause des serveurs de stockage gratuits et de l'accès internet rapide, le partage de vidéos est devenu très populaire. Selon Instagram, ce sont en moyenne 95 millions de photos et vidéos qui sont partagés chaque jour. C'est pourquoi la reconnaissance et la compréhension des vidéos, et la recherche de posts sont nécessaires.

4.3 Le geste

Dans l'article [11] il est souvent précisé que NAO prend en compte tout le corps du téléopérateur, et bouge lui aussi tout entier. On peut donc s'interroger sur ce qu'est un geste et ce que cela implique. Un geste est un mouvement des parties du corps employé pour signifier quelque chose. C'est pour cela que la reconnaissance des gestes présente l'un des nombreux défis posés dans le domaine de la vision par ordinateur et dans le domaine des interactions homme-robot. On peut déterminer deux catégories de gestes : statique et dynamique. Un geste statique correspond à la disposition du corps dont on étudie le mouvement à un instant donné. Ce type de geste a été étudié par plusieurs chercheurs mais ce n'est pas à ce type de geste que nous allons nous référer. Le geste dynamique [36] correspond à un changement de cette disposition sur la durée. C'est cette catégorie qui intéresse le plus les chercheurs. En effet, lorsqu'un être humain interprète un mouvement ou un geste, il s'intéresse à l'ensemble. C'est donc la catégorie qui offre les interactions les plus naturelles. C'est donc à cela que l'on va particulièrement s'intéresser. Un système de reconnaissance des gestes dynamiques regroupe généralement trois étapes : la détection, le suivi et la reconnaissance.

4.4 Détection et suivi

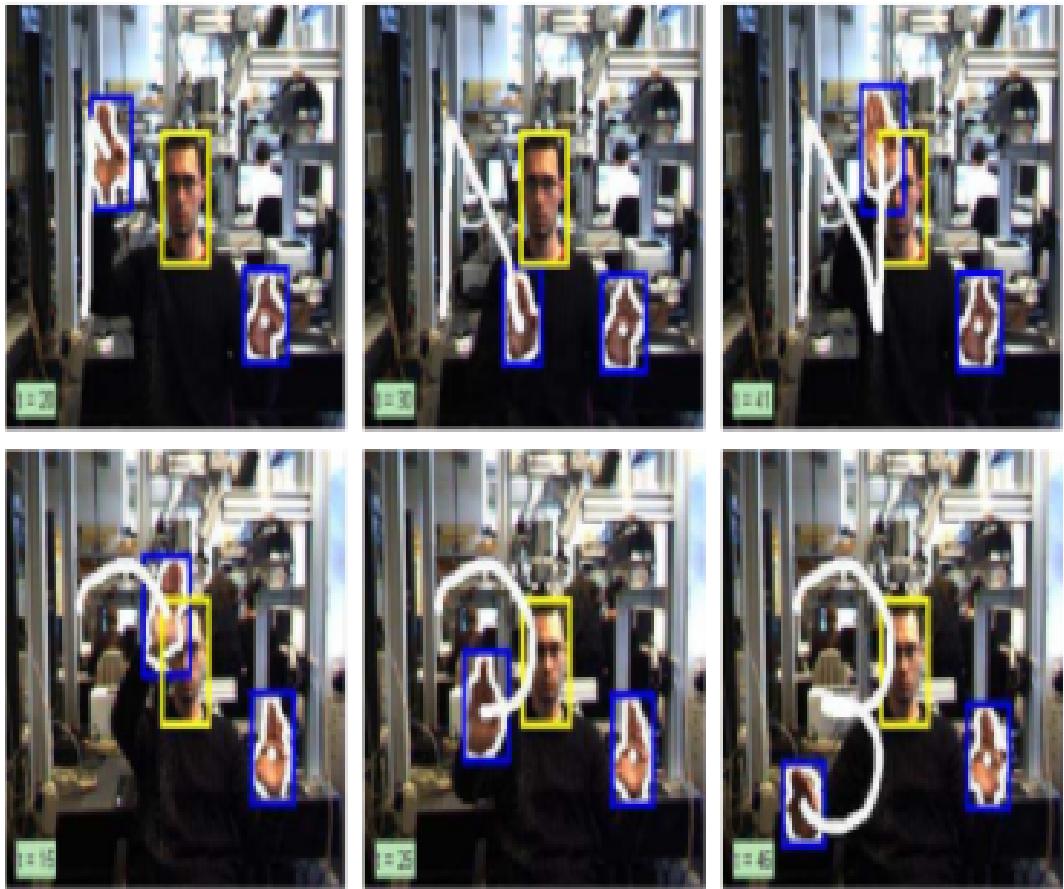
La première étape de l'analyse d'un geste dynamique est la détection. Pour cela, il existe diverses méthodes. La plus connue est liée à la couleur de la peau, comme [27], [12] ou encore [35] qui reconnaît à la fois la couleur de la peau et la couleur du vêtement de la personne détectée.



Reconnaissance de la couleur de la peau

Cependant, cette méthode nécessite un fond et un éclairage uniforme, ce qui rend cette méthode peu robuste dans l'absolu et en particulier dans le cas de gestes dynamiques. Il existe également d'autres méthodes qui sont basées sur la correspondance. Elles sont développées par exemple par [32], mais elles sont très sensibles à la rotation et au changement d'échelle ce qui les rend peu fiables pour la détection de mouvements dynamiques. C'est pourquoi les études les plus récentes se sont intéressées à la profondeur. Cette information est pertinente car elle permet d'ajouter une donnée aux mesures de correspondances, et de rendre cette détection plus stable.

Pour permettre l'acquisition de ce nouveau paramètre qu'est la profondeur, le domaine des capteurs 3D a connu une forte expansion. Parmi ces capteurs, nous retrouvons la caméra stéréoscopique pour le suivi des gestes. Elle est utilisée par [17] qui ont ainsi pu détecter le bout du doigt de l'utilisateur sur les deux images stéréo et ainsi reconnaître des lettres et des chiffres tracés dans l'air.



Reconnaissance de lettres tracées dans l'air

[14] ont également utilisé cette méthode. Dans ces images, les deux points sur lesquels le bout du doigt apparaît établissent une correspondance stéréo qui est utilisée pour évaluer la position du doigt dans l'espace 3D. Ainsi cette position est utilisée pour estimer la distance du doigt à partir de la table augmentée et, par conséquent, déterminer si l'utilisateur est en contact avec elle ou pas.



Table de réalité augmentée

Cette méthode a également été utilisée par [29] pour suivre et reconnaître les gestes des deux mains. Il leur a tout de même fallu résoudre le bruit horizontal généré par la caméra. En effet, elle est hypersensible à la lumière. Malgré tout, les systèmes stéréoscopiques fonctionnent uniquement pour des scènes texturées. Dans le cas d'objets ayant la même couleur que l'arrière-plan ou ayant la même texture, la segmentation ne se fera pas correctement. Une nouvelle famille de caméras 3D est donc apparue, les Swiss Ranger SR-2.



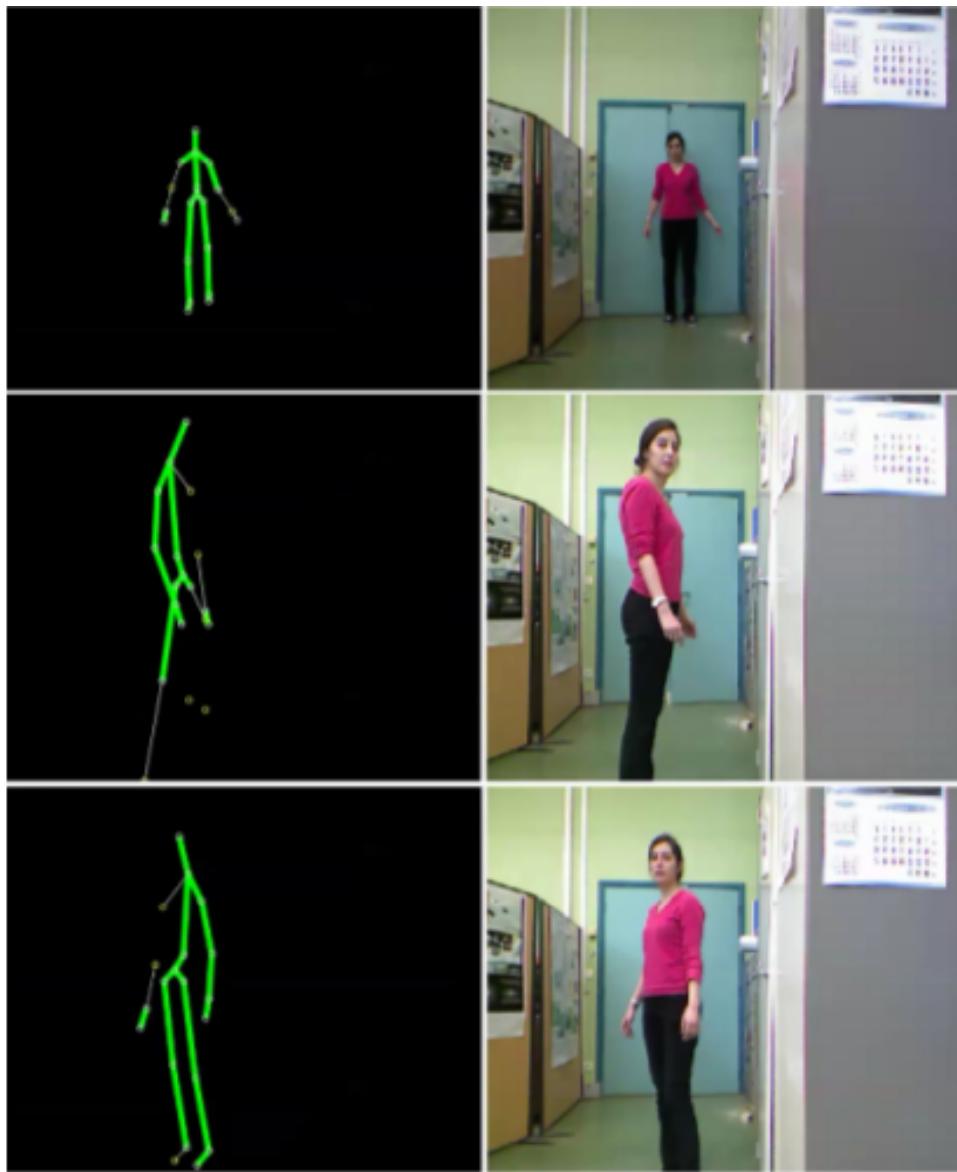
Caméra Swiss Ranger SR-2

C'est un capteur actif qui fournit des images de profondeur en temps réel. Contrairement aux caméras stéréoscopiques, contraintes à une taille minimale, ces nouvelles caméras peuvent être de taille beaucoup plus compacte. [24] et [19] ont utilisé ce capteur pour la reconnaissance des gestes des mains, basée sur l'estimation de mouvements 3D. Le mouvement peut être détecté par la différence entre deux gammes d'images, puis le signal est filtré avec un filtre passe-bande. Pour reconnaître le mouvement ainsi détecté, des descripteurs de contexte de forme harmonique ont été proposés. Il est ainsi possible de reconnaître les primitives de mouvements une fois le geste filtré. [16] ont choisi de transformer les données mesurées par la caméra en un nuage de points 3D. [4] ont développé une méthode pour reconnaître des gestes de pointage, en interprétant à la fois le mouvement et la cible du mouvement. Mais ce type de caméra a lui aussi ses limites : des interférences ont lieu lorsqu'il y a de la lumière directe comme la lumière du soleil. Cela brouille les informations de profondeur dans les régions directement exposées à la lumière. Un nouveau capteur de profondeur a donc tout naturellement été créé. Il a été intégré à la caméra *kinect* de Microsoft.



Caméra *kinect* de Microsoft

L'une des principales différences avec le modèle précédent est la résolution, qui est bien meilleure sur la *kinect* [9]. Cette nouvelle caméra a été utilisée pour de nombreux travaux de recherche. C'est le cas notamment de [7] pour le suivi des gestes des mains, de [20] qui étudient les gestes de pointage, ou encore de [30] suivent les mouvements de tout le corps. Certains chercheurs ont combiné l'information de profondeur avec la couleur pour réaliser une segmentation pour réaliser leur suivi. Par exemple, il existe un système de suivi en 3D de toutes les articulations de la main ainsi que la détection de sa position et de son orientation grâce au capteur *kinect*. Récemment, Microsoft a publié un kit de développement de tracking contenant l'algorithme "Skeleton". Cet algorithme permet de détecter et de suivre le corps humain complet grâce au capteur *kinect*. En effet, un squelette est projeté sur l'image du corps humain pour que chaque articulation réelle soit liée à une articulation du squelette projeté. On peut ainsi associer à chaque articulation du squelette un identifiant qui sera par la suite utilisé comme information par les algorithmes propres à chaque chercheur. [38], [5] et [22] ont par exemple utilisé "Skeleton" pour faire de la reconnaissance de gestes.



Représentation de "Skeleton"

Enfin, il existe l'ensemble de capteurs inertIELS XSens. Apparu en 2009, ce système permet le suivi en 3D des différentes parties du corps. Depuis, il a été utilisé pour divers travaux de recherche dans la télé-opération de robot [11], [28], [15] et notamment par [33] pour obtenir un modèle 3D de l'utilisateur et le faire reproduire au robot. Toutes ces recherches sont menées avec un robot NAO et en utilisant le logiciel MVN Analyze.

4.5 Extraction des données

L'objectif principal est d'obtenir le taux de reconnaissance des gestes le plus élevé possible. Pour cela, il faudrait trouver le vecteur caractéristique le plus approprié pour décrire un ensemble de gestes. Certains travaux comme [37] ont utilisé les Hidden Markov Model (HMM) pour la reconnaissance d'images en 2D. Une autre façon de combiner orientation, localisation et vitesse a été proposée par des chercheurs dans le but de reconnaître les caractères alphanumériques. D'autres chercheurs ont voulu tirer parti du suivi du squelette par exemple en choisissant comme vecteur caractéristique la distance de la main et du coude à la colonne vertébrale. D'autres ont quant à eux utilisé la méthode de Markov Caché pour reconnaître quatre gestes de commande afin de contrôler un robot. Pour cela, le vecteur caractéristique est le centre de la main détecté en 3D par la *kinect* et projeté par la suite dans l'espace 2D. [10] ont reconnu dix gestes choisis parmi les signaux visuels militaires de l'armée américaine. Cela permet de contrôler un robot mobile. La reconnaissance des gestes a été réalisée grâce à un système de classification réalisé par un réseau de neurones qui ont extrait deux types de caractéristiques : les rotations et les angles. Ces caractéristiques ont ensuite été comparées pour obtenir un taux de reconnaissance élevé. Cependant, la méthode de Markov cachée a aussi des limites, comme expliqué par [13]. Une autre méthode est la déformation temporelle dynamique pondérée, dont le but est de reconnaître deux gestes. Pour cela, deux caractéristiques ont été extraites pour le geste "salut" : la distance euclidienne entre l'articulation de la main, et l'angle fait par l'articulation du coude, et trois caractéristiques ont été extraites pour le geste "pointé" : angle fait par l'articulation du coude, la distance entre les articulations de la hanche et de la main, et la position de l'articulation de la main. Ce travail a pour but d'interagir avec le robot NAO.



NAO imite les gestes qu'il reconnaît

D'autres chercheurs [21] ont quant à eux choisi d'intégrer l'angle d'Euler pour reconnaître les gestes de l'utilisateur afin de contrôler un robot.

4.6 Reconnaissance de gestes

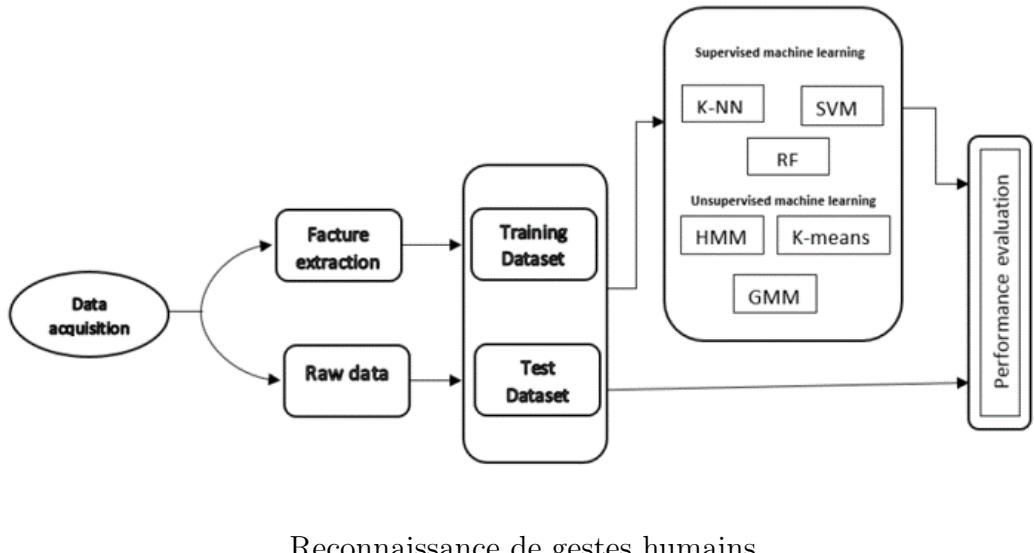
La reconnaissance de gestes à partir des données visuelles a été largement utilisée pour de nombreuses tâches et compte aujourd’hui quelques applications. La littérature associée divise la reconnaissance des gestes selon deux approches principales : la mise en correspondance de modèles statiques et les modèles dynamiques. Dans le premier cas, on parle de reconnaissance de configuration, ou parfois de "gestes statiques". Mais c'est au second cas que l'on va s'intéresser. Différentes méthodes ont été exploitées pour ce type de reconnaissance. Par exemple, les réseaux de neurones artificiels (RNA) ont été utilisés pour cela. Ils sont inspirés de la méthode de travail du cerveau humain qui est totalement différente de celle d'un ordinateur. Un réseau de neurones artificiel représente une boîte noire, d'où la difficulté de paramétrage [25]. De même, la partie apprentissage est délicate, car le choix des valeurs de poids de connexion est compliqué. Dans le cas de bases de données, les réseaux de neurones ne permettent de traiter les cas avec des attributs symboliques qu'après un encodage adapté. En revanche, d'autres techniques d'apprentissages telles que les SVMs et les arbres de décisions ne nécessitent pas d'encodage spécifique. C'est pour cela que de nombreux travaux récents comme [18] ou [1] favorisent d'autres méthodes au détriment des réseaux de neurones. La plus utilisée est la méthode des machines à vecteurs de support. Il s'agit d'une technique d'apprentissage statistique qui consiste à projeter les données de l'espace d'entrée non-linéairement séparables dans un espace de plus grande dimension appelé "espace de caractéristiques", de façon à ce que les données deviennent linéairement séparables. L'idée clé est donc d'augmenter la dimension de la représentation de ces données, et ensuite de calculer la surface de séparation entre les classes dans cet espace de représentation. Il reste cependant une grosse difficulté : les vecteurs descripteurs décrivant deux classes aux propriétés spatiotemporelles similaires sont très difficiles à discriminer, et la variabilité accroît d'autant la confusion. En effet, les SVMs sont bien adaptés à la classification binaire, mais dans le cas contraire où il s'agit de N classes, on doit utiliser l'approche un contre-un. Cette méthode permet de créer des classifieurs spécialisés dans la comparaison "classe à classe". Donc pour une classification de N classes on aura $N*(N-1)/2$ classifieurs. Il y a donc une forte augmentation de sa complexité avec le nombre de classes, étant donné que pour N classes, il y a $N*(N-1)$ comparaisons. Par ailleurs, cette méthode est peu efficace en cas de classification de très petite dimension ; elle ne donne pas les résultats escomptés. D'autres travaux ont utilisé une méthode d'apprentissage différente : la déformation temporelle dynamique [31]. Cet algorithme permet de trouver un appariement optimal entre deux séquences en mesurant la similarité entre deux suites qui peuvent varier au cours du temps. Le principe est de trouver un chemin selon certaines règles pour minimiser l'ensemble des distances entre les vecteurs des deux séquences. Cette méthode est utilisée pour des tâches à petit vocabulaire, mais elle a un champ d'application relativement limité de par ses défauts. En effet, les applications de reconnaissance des gestes continus, la déformation temporelle dynamique est peu précise parce qu'elle nécessite une grande puissance de calcul en raison de sa complexité [23]. Une autre méthode a donc été développée. Il s'agit des modèles de Markov Cachés, déjà évoqués précédemment et qui sera développée par la suite. Cette méthode a connu un grand succès dans divers domaines d'application. Elle a été initialement introduite dans la reconnaissance vocale, puis plus tard dans le domaine de la reconnaissance

gestuelle avec par exemple [34]. Enfin, cette méthode a été utilisé dans les deux domaines simultanément [26].

4.6.1 Machine Learning

Le machine learning est défini comme concevoir une fonction de mappage à partir d'un jeu de données de gestes pour obtenir une classification des gestes. Jusqu'à maintenant, plusieurs méthodes de machine learning ont été étudiés [6], pour définir une fonction depuis une instance de geste, vers des classes de gestes. Depuis les classes sont connues (les données sont « trained », c'est-à-dire transformées), la reconnaissance est dans ce travail implémentée par des techniques d'apprentissage supervisé, ce qui signifie qu'un superviseur entraîne notre base de données de gestes (clustering et classification), décrivent l'instance des gestes (extraction de caractéristiques) et les informations de la classe correspondante pour chaque instance du geste (classification). Par la suite, la fonction de mappage est construite à partir de l'instance des gestes et les informations des classes. La reconnaissance d'une instance inconnue d'un geste (donnée non supervisée) peut être réalisée en utilisant la fonction de mappage disponible.

Donc le processus de reconnaissance de geste inclus les étapes suivantes : représentation de l'action humaine (extraction de caractéristiques), clustering (regroupement), et classification des gestes de l'utilisateur. Ceci est détaillé sur la figure suivante :



Reconnaissance de gestes humains

4.6.2 Clustering

Le clustering est une technique de machine learning qui regroupe des points donnés obtenus par des méthodes d'extraction de caractéristiques. C'est une méthode d'analyse statistique de données, d'apprentissage non supervisé. En donnant un ensemble de données, on peut utiliser un algorithme de clustering pour classer chaque donnée dans un groupe spécifique. Cela signifie que les données disposant de propriétés ou de caractéristiques similaires seront dans le même groupe, tandis que des

données qui auront de fortes différences ne seront pas dans le même groupe.

Dans notre domaine, le clustering est utilisé pour obtenir des informations précieuses à partir de données dérivées en reconnaissant les groupes auxquels les données appartiennent lorsque nous appliquons un algorithme de clustering.

Clustering K-Means

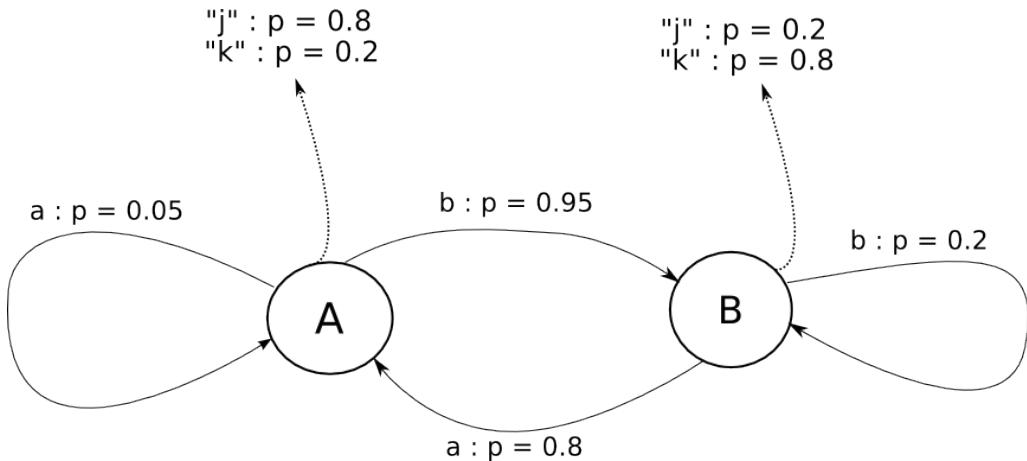
K-Means est probablement le plus connu des algorithmes de clustering. Cet algorithme a de nombreuses applications dans différents domaines de l'analyse de données et le machine learning suit les étapes suivantes :

- 1) Pour commencer, on choisit un nombre, K, qui correspond au nombre de classes/groupes à réaliser et leurs points centraux sont initialisés aléatoirement. Le nombre optimal de clusters peut être déterminé par des méthodes de tests statistiques et les points centraux doivent être de même longueur.
- 2) Chaque donnée est regroupée en calculant la distance euclidienne entre cette donnée et les points centraux de chaque groupe, puis la donnée est placée dans le groupe dont le centre est le plus proche.
- 3) A partir de ces groupes de données, le centre du groupe est recalculé en prenant la moyenne de toutes les données du groupe.
- 4) On répète ces étapes, soit pour un nombre défini d'itérations, soit jusqu'à ce que les centres de groupe ne varient pas ou très peu entre deux itérations.

Comme tout est calculé grâce à la distance entre les données et les centres des groupes, le K-Means a l'avantage d'être rapide, avec une complexité linéaire en $O(n)$. Mais comme l'algorithme K-Means commence par un choix aléatoire des centres, il peut y avoir des résultats différents lors de deux exécutions du même programme avec le même jeu de donnée.

Clustering HMM

La méthode de clustering Hidden Markov Model (HMM) est un modèle statistique dans lequel le système modélisé est supposé être un processus markovien de paramètres inconnus et qui prend en entrée une matrice contenant les données à traiter.



Exemple d'automate représentant un modèle de Markov

Un automate de HMM est un quadruplet $\{S, \Pi, A, B\}$ des ensembles décrits sui-

vant :

- S_i l'état i ;
- π_i la probabilité que S_i soit l'état initial ;
- a_{ij} la probabilité de la transition $S_i \rightarrow S_j$;
- $b_i(k)$ la probabilité d'émettre le symbole k étant dans l'état S_i ;

Sous contrainte :

- $\sum_i \pi_i = 1$ la somme des probabilités des états initiaux est égale à 1 ;
- $\forall i, \sum_j a_{ij} = 1$ la somme des probabilités des transitions partant d'un état est égale à 1 ;
- $\forall i, \sum_k b_i(k) = 1$ la somme des probabilités des émissions partant d'un état est égale à 1.

On génère ensuite un HMM initial, de la forme $\lambda(A, B, \pi)$. On commence par choisir arbitrairement deux entiers, N et M. Pour optimiser les résultats de la reconnaissance, il est important de répéter le processus avec diverses valeurs de N et M. Pour chacune des matrices A, B et π , la somme des éléments de chaque ligne de chacune de ces matrices vaut 1. A est de taille [NxN], B de taille [NxM] et π de taille [1xN]. Tous les éléments de ces matrices sont générés aléatoirement en respectant la condition de somme. On utilise à présent l'algorithme de Baum Welch. Cet algorithme prend en entrée la matrice BD et le système HMM initial. Il sort un système HMM optimisé, avec les matrices A, B et π optimisées à notre problème. C'est ce système λ_{opt} qui servira à reconnaître les mouvements inconnus.

Cet algorithme est énormément utilisé dans le domaine de la reconnaissance, que ce soit vocale, gestuelle ou encore d'écriture.

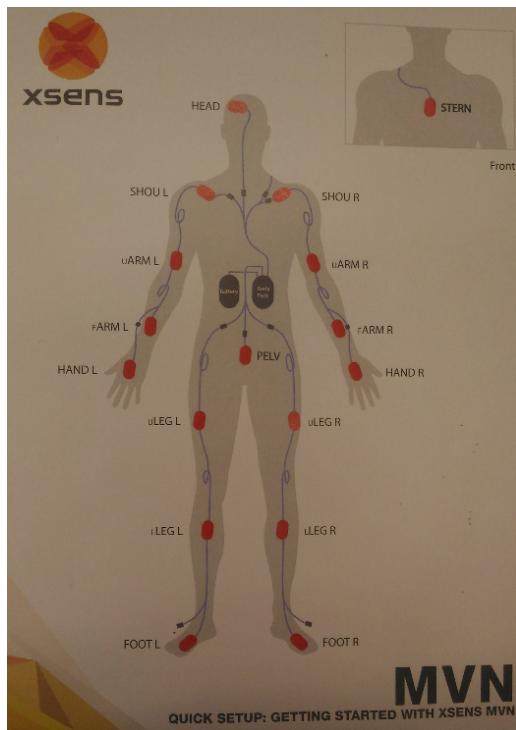
C'est cet algorithme qui a été retenu pour notre travail car moins variable que le K-Means. L'utilisation que nous en avons faite est décrite par la suite.

5 Réalisation de la base de données

5.1 Utilisation de XSens et MVN Analyze

Mise en place du matériel

La première étape a été de prendre en main la combinaison XSens Awinda et le logiciel associé. En effet, ce matériel n'avait encore jamais été utilisé à l'IBISC. La partie hardware du XSens Awinda est composé d'une chemise qui doit être la plus ajustée et moulante possible, de neuf bandes de velcro, de deux mitaines et d'un bandeau. Tout cela permet de placer les 17 capteurs partout sur le corps de la personne dont on veut obtenir les mouvements : un sur chaque bande de velcro, un dans chaque mitaine, un dans chaque chaussure, un dans le bandeau, et trois sur la chemise. Un dix-huitième capteur est disponible pour pouvoir obtenir les mouvements d'un autre objet, comme d'une épée ou d'une baguette magique que tiendrait le modèle dont on capture les mouvements.

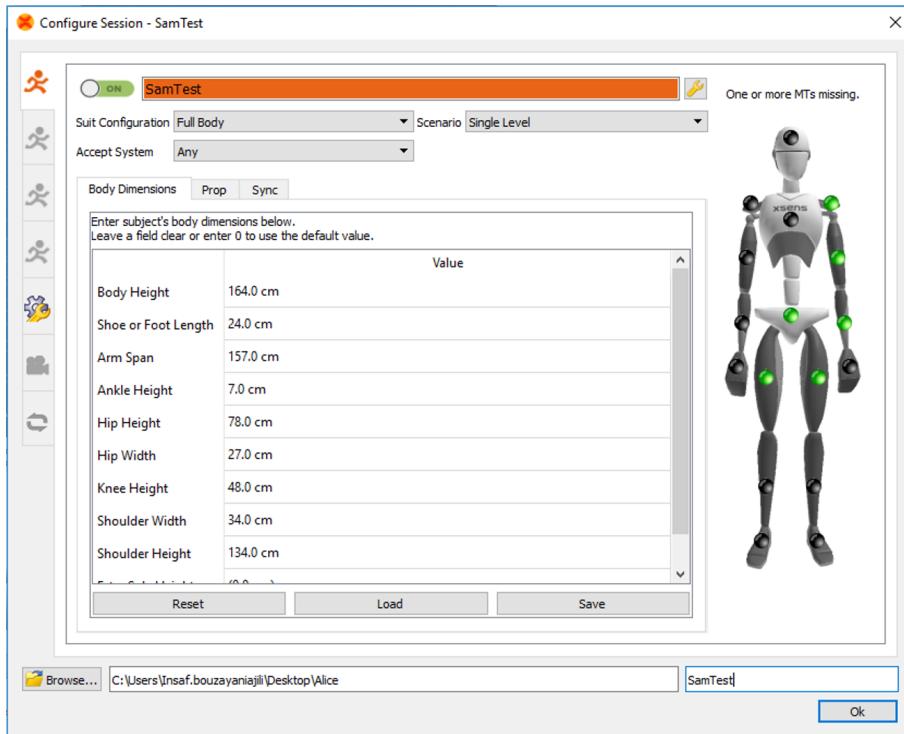


Emplacement des différents capteurs

Le logiciel utilisé pour l'enregistrement et le traitement des données est MVN Analyze. Pour commencer, il faut brancher la clé USB contenant la licence à l'ordinateur. Il faut ensuite prendre les mesures précises de la personne qui va s'équiper des capteurs XSens. Lorsqu'on lance le logiciel, il faut commencer par créer une session, ou ouvrir une session existante.

Ouvrir une session

Lorsque l'on ouvre le logiciel la première fois ou lorsque l'on souhaite modifier les réglages de session, la fenêtre suivante apparaît.



Ecran de réglages de session

Si l'on souhaite ouvrir une nouvelle session, il faut la nommer (champ en orange sur l'image), puis renseigner les mesures de la personne qui va être suivie. Il est aussi possible de compléter ces valeurs à la main, ou de charger un fichier de mesure déjà existant. Si l'on a renseigné les mesures à la main, il est conseillé d'enregistrer le nom de la personne et ses mesures pour gagner du temps lors des utilisations futures.

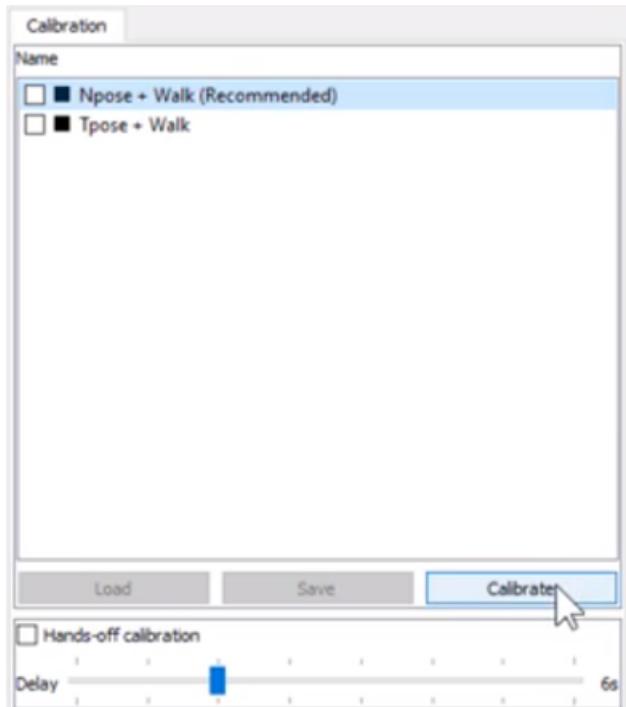
Le champ "Suit Configuration" permet de choisir de ne suivre que certaines parties du corps. Le champ "Scenario" permet de définir si la capture qui va être effectuée est réalisée sur un seul niveau de hauteur, au niveau du sol ou sur plusieurs niveaux (c'est le cas par exemple si le modèle monte un escalier). Enfin, le champ "Accept System" permet de déterminer à quel type de produit XSens on souhaite se connecter. Il est conseillé de choisir "any", ce qui permettra au logiciel de détecter les capteurs, quelle que soit la version. Dans notre cas, on pourrait également sélectionner "Awinda".

Il faut ensuite que tous les capteurs sélectionnés soient détectés. Sur l'image ci-dessus, les deux capteurs des cuisses, celui du pelvis, ainsi que ceux de l'épaule, du biceps et de l'avant-bras gauche sont détectés. Il peut arriver que certains capteurs soient détectés comme appartenant à un autre ensemble de capteurs. Il faut alors les resynchroniser en appuyant sur le petit bouton physique des capteurs qui ne sont pas correctement détectés.

Il suffit alors de cliquer sur "ok" en bas à droite. Un squelette recroquevillé s'affiche alors à l'écran s'il n'y a pas encore eu de calibrage.

Calibrage

Une fois que l'on a ouvert une session, on peut réaliser un calibrage.



Fenêtre de calibrage

Il existe deux types de position que l'on peut adopter : en T (bras ouverts à l'horizontal, dos droit, pieds proches et parallèles), ou en N (les bras le long du corps, dos droit, pieds proches et parallèles). La position en N, plus facile à tenir et à reprendre à l'identique, est conseillée.



Position N

Pour réaliser une calibration, il faut sélectionner la pose, que la personne à calibrer se mette en position, et que quelqu'un d'autre clique sur "Calibrate". La personne équipée de XSens doit garder la pose environ quatre secondes, puis avancer à allure modérée de quatre ou cinq pas en accompagnant le déplacement d'un léger mouvement de bras, faire demi-tour, faire le même nombre de pas en sens inverse,

faire à nouveau demi-tour et reprendre sa position initiale (T ou N) à la place initiale. Il est nécessaire de rester statique jusqu'à ce qu'on ait appliqué la calibration. Pour que la calibration soit appliquée, il suffit d'attendre jusqu'à ce qu'une fenêtre s'ouvre pour demander si l'on veut recommencer ou appliquer la calibration. Le logiciel donne également un avis sur la qualité de la calibration, mais il est toujours possible d'appliquer puis de constater si la qualité nous convient ou non.

Il se peut qu'un capteur se détache lorsque des mouvements sont effectués. Il est alors nécessaire de refaire une calibration.

5.2 Acquisition de données

Il a ensuite fallu enregistrer diverses séquences de mouvements selon le protocole que nous avons établi.

Pour chaque individu, nous avons relevé le prénom, l'âge et le sexe, puis nous avons pris ses mesures pour pouvoir utiliser le logiciel MVN Analyze. Il faut ensuite équiper chaque individu des 17 capteurs XSens, puis les calibrer. Lors de la calibration, l'axe des X est déterminé. Pour que les données soient exploitables, nous avons tracé l'axe des X à suivre au sol, et nous avons replacé le squelette à l'origine pour chaque extrait vidéo. Ainsi, les données sont exploitables directement puisqu'elles se situent toutes dans le même repère.

Pour chaque personne, nous avons montré le premier mouvement, qui dit au robot d'avancer. Cette première version, "normale", correspond à l'émotion "calme". Nous avons ensuite montré ce même mouvement avec les émotions "joyeux", "triste" et "en colère". La personne a ensuite réalisé cinq fois le mouvement avec l'émotion "joyeux", puis cinq fois avec l'émotion "triste" puis cinq fois avec l'émotion "en colère" et enfin cinq fois calmement. Puis selon le même schéma, nous avons enregistré le geste qui dit de tourner à droite, celui qui dit de saluer, celui qui dit de s'arrêter et enfin celui qui dit de danser. Pour chaque individu, nous avons donc obtenu 100 vidéos. Nous avons demandé à 11 personnes de réaliser ces mouvements, obtenant ainsi 1100 vidéos.

Une fois toutes ces vidéos enregistrées au format .mvn, il a fallu les rendre exploitables à la fois en pouvant afficher les données sous excel, et en rendant les vidéos visibles sur n'importe quel ordinateur. En effet, les fichiers d'extension .mvn ne peuvent pas être ouverts sans disposer du logiciel MVN Analyze et d'une licence valide. Tous les fichiers ont donc été ouverts à nouveau puis exportés deux fois. Une première fois au format .mvnx, lisible par Excel, et une seconde fois au format .mp4.

5.3 Analyse des résultats

5.3.1 Analyse mathématique

Pour réaliser l'analyse mathématique, mon seul travail a été de faire en sorte que les données enregistrées précédemment puissent être ouvertes sous Excel. Pour cela, il a fallu ouvrir puis exporter les 1100 enregistrements, pour que l'extension devienne .mvnx. Ces fichiers sont alors directement exploitables sous Excel. Mes collègues avaient déjà réalisé un algorithme mis en place grâce au logiciel Matlab

pour analyser toutes les données. Les fichiers d'extension .mvnx se sont avérés lisibles par Matlab comme prévu, ce qui a permis de réaliser la partie machine learning et de déterminer les caractéristiques importantes de chaque émotion. Par exemple, pour la colère, nous pensions qu'il y avait deux caractéristiques majeures : la vitesse et l'amplitude. Dans les faits, la caractéristique la plus marquée est l'accélération.

5.3.2 Etude statistique

Le principe de tout ce travail est de faire en sorte que le robot réagisse en fonction des émotions qu'il détecte. Nous avons donc demandé à nos premiers volontaires de réaliser 100 mouvements avec des émotions. Cependant, nous les avons guidés dans ces émotions pour pouvoir avoir des ressemblances suffisantes d'un enregistrement à l'autre et pouvoir faire du machine learning. Or il est essentiel que ces mouvements restent emprunts d'émotions discernables par l'humain. Pour vérifier cela, nous avons réalisé deux protocoles expérimentaux.

Dans le premier protocole, on proposait à un volontaire un ensemble de vingt vidéos, chacune représentant un des cinq gestes avec l'une des quatre émotions. La personne volontaire devait regarder les vidéos une par une et pour chacune dire quelle émotion était représentée. La réponse était enregistrée, puis on donnait la bonne réponse et la personne volontaire devait noter la vraie émotion de 1 à 5. L'échelle est la suivante : 1 si l'émotion est totalement incohérente avec la vidéo, 2 si elle ne semble pas vraiment compatible, 3 si on n'en sait rien, 4 si l'émotion pourrait concorder avec la vidéo et 5 si maintenant qu'on sait quelle émotion a été théoriquement engregistrée, c'est évident. Pour réaliser les dossiers de vidéos, nous avons sélectionné arbitrairement des vidéos de chacun des 11 participants, une pour chaque émotion de chaque mouvement. Nous avons réalisé comme cela plusieurs dossiers. Nous avons pris cinq volontaires sur ce protocole. Mais d'une part, en mélangeant les mouvements et sans pouvoir les comparer les uns aux autres, les résultats étaient vraiment très mauvais : peu de gens arrivaient à reconnaître les émotions. De plus, comme chacun passait sur des dossiers de vidéos différents, on ne pouvait pas faire de statistiques regroupées pour vérifier si la personne répondait au hasard ou si l'une de nos vidéos était particulièrement mauvaise. Enfin, nous ne pouvions pas tester suffisamment systématiquement les vidéos. Nous avons donc changé de protocole.

Lors de la réalisation du second protocole, nous avons cherché à faire évaluer un maximum de vidéos différentes, par le plus de monde possible. Pour cela, nous avons créé onze groupes de vidéos, ce qui correspond aux onze volontaires de la partie "enregistrement de mouvements". Dans chaque dossier se trouvent 5 autres dossiers correspondant aux 5 mouvements et dans chacun de ces dossiers se trouvent huit vidéos numérotées de 1 à 8. Ces huit vidéos correspondent à deux fois chaque émotion d'une même personne. Ces deux vidéos ont été sélectionnées aléatoirement parmi les cinq initialement réalisées. Pour chacun des onze groupes, nous avons demandé à dix personnes de visionner les quarante vidéos. Il nous fallait donc cent-dix volontaires. Pour simplifier le travail de ces volontaires, nous avons créé un document papier à remplir, comme suit.

Nom et prénom:

Age:

Dance	Happy	Angry	Sad	Calm
1				
2				
3				
4				
5				
6				
7				
8				

Move	Happy	Angry	Sad	Calm
1				
2				
3				
4				
5				
6				
7				
8				

Salut	Happy	Angry	Sad	Calm
1				
2				
3				
4				
5				
6				
7				
8				

Stop	Happy	Angry	Sad	Calm
1				
2				
3				
4				
5				
6				
7				
8				

Turn	Happy	Angry	Sad	Calm
1				
2				
3				
4				
5				
6				
7				
8				

Document à remplir pour l'évaluation

Dans chaque case, la personne devait noter entre 1 et 5 l'émotion, 1 était je suis sûr que ce n'est pas ça, 2 je pense que ce n'est pas ça, 3 je n'en sais rien, 4 je pense que c'est cette émotion et 5 je suis sûr que c'est cette émotion. Les volontaires avaient le droit de regarder toutes les vidéos d'un même mouvement avant de mettre leurs notes. De plus ils étaient prévenus que chaque émotion était représentée deux fois pour chaque mouvement. Pour obtenir suffisamment de volontaires, nous sommes allés voir différents professeurs de l'université pour leur demander si leurs élèves pouvaient passer une partie de leurs pauses lors des TPs à regarder nos vidéos. Le temps d'explication et de notation était en moyenne de 15 minutes et pour gagner du temps, nous mettions les vidéos directement sur les ordinateurs des étudiants et tous pouvaient faire l'évaluation en même temps.

Une fois toutes les évaluations terminées, il a fallu retranscrire les résultats sous Excel, puis ils ont été exploités via un algorithme sous Matlab, réalisé par mes collègues.

6 Méthode de reconnaissance de gestes

Pour reconnaître un geste, il y a deux principales étapes. La première est l'apprentissage. En effet, il faut enregistrer une base de données composée de différents gestes, puis la traiter pour obtenir des valeurs de référence. La seconde étape est la reconnaissance à proprement parler. Il s'agit de traiter un geste inconnu et de comparer les valeurs obtenues avec les valeurs de référence pour trouver à quel geste ce nouveau geste ressemble le plus.

6.1 Apprentissage

L'apprentissage se découpe en quatre parties. La première étape est l'acquisition des données brutes qu'il faudra traiter. Vient ensuite l'extraction des caractéristiques à partir des données, pour obtenir un vecteur descripteur pour chaque répétition de chaque geste. Chaque répétition de chaque geste est ensuite échantillonnée. En effet, chaque répétition de chaque geste a à priori une durée et donc un nombre de frame différent. Il s'agit donc ici de normaliser la taille des vecteurs obtenus. Chaque vecteur est ensuite transformé en une valeur caractéristique. Vient enfin la quantification, qui permet de regrouper les gestes entre eux. Ce sont ces étapes qui vont maintenant être détaillées.

6.1.1 Acquisition des données

Quel que soit le contexte, la première étape est l'acquisition des données. Cela peut aussi bien se faire avec une *Kinect* ou avec un système comme XSens. Dans notre cas, l'acquisition a été faite avec une *Kinect* version 2. On obtient des données brutes grâce au package *openni_tracker*. On obtient ainsi pour chaque articulation la position en x, y et z, ainsi que leur rotation en x, y et z. Il faut ensuite convertir le fichier des données brutes, qui est dans un format spécifique au mode d'acquisition, en fichier d'extension .csv. C'est ce fichier qui est la sortie de cette étape et qui servira d'entrée à l'étape suivante.

6.1.2 Extraction de caractéristiques

A partir des données brutes, il nous faut extraire des données caractéristiques. Il existe divers algorithmes pour obtenir un vecteur descripteur, contenant diverses informations. Dans notre cas, nous utilisons le modèle LMA. Grâce à cet algorithme, nous avons des descripteurs répartis en quatre catégories : Corps, Forme, Effort et Espace. La catégorie Corps contient toutes les caractéristiques physiques. La catégorie Forme contient trois descripteurs, Mise en forme (ce qui correspond au volume occupé par la personne), Mouvement direct (cela correspond à un nombre entre 0 et 1 qui est le résultat du quotient $\frac{\text{PositionDepart} - \text{PositionArrivée}}{\text{CheminParcourru}}$) et Flux de forme (modification du volume). La catégorie Effort contient quatre descripteurs, Poids (ce qui correspond à l'accélération du mouvement), Temps (cela correspond à la vitesse du mouvement), Flux (si le mouvement est fluide ou non), et Espace (orientation du mouvement selon trois axes). Enfin, la catégorie Espace correspond au vecteur

normal au plan formé par les deux épaules et le centre du bassin. Toutes ces transformations ont été implémentées sous Matlab, et donc pour chaque mouvement, on obtient un vecteur descripteur. Dans notre cas, on a 47 descripteurs car on ne tient pour l'instant pas compte de la composante Effort. Ce sont ces vecteurs qui seront ensuite échantillonnés et qui donc serviront d'entrée à l'étape suivante.

6.1.3 Echantillonnage

Pour chaque geste, il n'est pas évident que l'on ait le même nombre de frames. C'est pour cela qu'un algorithme a été créé pour que chaque geste ait un nombre choisi de frames. On a donc, pour chaque geste, une matrice de 47 colonnes (ce qui correspond au nombre de descripteurs) et de T lignes où T correspond au nombre de frame choisi précédemment. Ce sont ces matrices qui serviront d'entrée pour l'étape suivante.

6.1.4 Quantification

A partir de la matrice de chaque geste, une transformation de chaque ligne est effectuée pour n'avoir plus qu'une valeur. On obtient donc un vecteur, dont chaque valeur correspond aux caractéristiques d'une ligne de l'ancienne matrice. Si on écrit ce vecteur de longueur T horizontalement pour chaque geste et que l'on place ces vecteurs les uns en-dessous des autres, on obtient une matrice NxT où N est le nombre de gestes de la base de données et T le nombre de frames conservées. Nous appellerons cette matrice BD.

On applique à cette matrice la méthode Hidden Markov Model (HMM). Pour faire cela, on génère un HMM initial, de la forme $\lambda(A, B, \pi)$. On commence par choisir arbitrairement deux entiers, N et M. Pour optimiser les résultats de la reconnaissance, il est important de répéter le processus avec diverses valeurs de N et M. Pour chacune des matrices A, B et π , la somme des éléments de chaque ligne de chacune de ces matrices vaut 1. A est de taille [NxN], B de taille [NxM] et π de taille [1xN]. Tous les éléments de ces matrices sont générés aléatoirement en respectant la condition de somme. On utilise à présent l'algorithme de Baum Welch. Cet algorithme prend en entrée la matrice BD et le système HMM initial. Il sort un système HMM optimisé, avec les matrices A, B et π optimisées à notre problème. C'est ce système λ_{opt} qui servira à reconnaître les mouvements inconnus.

6.2 Reconnaissance

6.2.1 Traitement des données

Le nouveau geste à reconnaître est tout d'abord enregistré sous forme de données brutes. On en extrait ensuite un vecteur descripteur sous Matlab, exactement de la même manière que pour tous les gestes de la base de données, puis on échantillonne ce vecteur pour qu'il ait une longueur T comme les autres. De la même manière que pour chaque geste précédent, pour chaque frame de ce mouvement inconnu on extrait une valeur unique et on obtient ainsi un vecteur θ de longueur T. Le vecteur pourra enfin être comparé à la base de données.

6.2.2 Comparaison

On utilise un algorithme de Forward-Backward pour calculer $P(\theta|\lambda_{opt})$. L'algorithme calcule la probabilité que le geste soit similaire à chacune des N catégories créées par l'algorithme Baum Welch (N étant l'entier choisi arbitrairement précédemment). La probabilité la plus importante permettra de déterminer à quels gestes ce nouveau geste ressemble.

C'est cette méthode qui est utilisée pour reconnaître les gestes par le robot NAO en temps réel, et qui permettra notamment de reconnaître les émotions, mais pour cela ce seront 85 descripteurs et non plus 47 qui seront utilisés. De plus, la méthode de reconnaissance utilisée sera random forest. A ce jour, Insaf AJILI travaille à la mise en place des algorithmes sur notre base de données.

6.3 Résultats

Les programmes développés par Insaf AJILI, doctorante avec laquelle j'ai travaillé, ont permis de donner les résultats suivants.

Gestures	Accuracy (%)	Precision (%)	Recall (%)	F-score
Waving	92.03	93.56	93.04	0.93
Moving	80.91	78.98	79.50	0.80
Dancing	76.34	76.71	74.72	0.75
Stopping	82.17	83.64	83.63	0.84
Pointing	88.08	89.17	88.51	0.91
All gestures	83.18	83.37	83.68	0.83

Tableau des résultats de la reconnaissance de geste

Avec : Precision = $\frac{NBGestesCorrectementPlacesDansLaCategorie}{NBGestesPlacesDansLaCategorie}$, Recall = $\frac{NBGestesCorrectementPlacesDansLaCategorie}{NBDeGestesAppartenantALaCategorie}$ et F-score = $2 * \frac{Precision * Recall}{Precision + Recall}$. Le taux de reconnaissance et les résultats de manière générale sont satisfaisants.

Cependant, lors de la reconnaissance de gestes, l'émotion n'était pas prise en compte. Il faudra mettre en place les 85 descripteurs au lieu des 47 pour pouvoir tester la reconnaissance des émotions.

7 Liaison XSens-NAO

Après la création et l'exploitation de la base de données, j'ai pris connaissance du travail qui avait déjà été réalisé pour la reconnaissance de gestes. J'ai donc lu les rapports et articles précédents, puis j'ai cherché à intégrer XSens à ce qui existait.

7.1 Situation initiale

Une version utilisant la *kinect* existait déjà. Pour la remplacer par *XSens*, il est nécessaire de changer le mode d'acquisition des données, puisque leur traitement reste le même. Pour commencer, la version existante a été testée, pour la comprendre et j'ai lu tout le code pour essayer de déterminer où se faisait l'acquisition de données. Malheureusement, ce n'était pas seulement une simple ligne de code à modifier. En effet, pour utiliser la *kinect* avec ROS, le package *openni_tracker* est utilisé. Or ce package permet d'avoir directement les coordonnées de la plupart des éléments du corps détectés enregistrés dans les variables head, neck, torso, left_shoulder, left_elbow, left_hand, right_shoulder, right_elbow, right_hand, left_hip, left_knee, left_foot, right_hip, right_knee et right_foot. J'avais commencé par installer le driver *ethzasl_xsens_driver* pour détecter l'ensemble des capteurs XSens. Cependant, ce driver n'était en fait pas destiné à la combinaison complète de capteurs, mais à l'acquisition de données depuis un seul capteur inertiel. De plus, la combinaison Awinda que nous possédons dispose de 17 capteurs et chacun est dédié à une partie du corps spécifique. Le driver est donc incompatible avec le matériel disponible à l'IBISC.

7.2 Réseau

Pour régler ce problème, j'ai fait appel à Felix WOLBERT, qui nous a fourni une assistance pour pouvoir utiliser XSens avec le logiciel déjà existant. Il m'a dit que le logiciel MVN Analyze permet de créer un serveur UDP, ce qui permet d'obtenir les données des centrales inertielles en temps réel. Il est également possible de créer un serveur TCP, mais il est recommandé dans la documentation d'utiliser le processus UDP. Dans notre cas, cela nous semble cohérent car pour reconnaître des mouvements en temps réel, il est préférable de favoriser la rapidité de la transmission des données, même si le risque est de perdre quelques données. Il a donc fallu créer un client en C++ (langage du logiciel déjà existant). Pour commencer, j'ai lu la documentation pour pouvoir démarrer le serveur avec MVN Analyze. J'ai commencé par écrire le client UDP sous linux, et également un serveur UDP sous linux pour pouvoir tester mon client. Le premier cas réalisé a été un serveur qui attend et affiche le message qu'il reçoit puis se ferme. Le client envoie simplement un message "plop". Cette première étape permet de vérifier que le client fonctionne et que s'il y a un problème, il ne vient pas du code lui-même.

Le PC sur lequel MVN Analyze est installé dispose un dual boot linux/windows. J'ai donc essayé de simplement déplacer le serveur sur l'autre ordinateur, sous linux pour changer le moins de paramètres possible. Les premières tentatives étant un échec, j'ai essayé d'envoyer un message ping simplement les machines entre elles. Mais cela n'a pas fonctionné. J'arrivais à ping dans un sens mais pas dans l'autre.

Il s'est avéré que la machine en dual boot était en réseau filaire, et l'autre en wifi. Or les deux réseaux étant distincts, il est très compliqué de communiquer entre les deux machines. La machine en wifi disposait bien à ce moment-là d'un port réseau, mais l'utilisation du robot NAO nécessite impérativement d'être branché en filaire à la machine pour avoir une stabilité suffisante. Après avoir ouvert la machine avec l'aide du responsable réseau, il est apparu que la machine disposait d'une carte réseau filaire, d'une carte réseau wifi et aucun port de libre. Il a été décidé de remplacer la carte wifi par une seconde carte réseau. Le ping a alors fonctionné dans les deux sens.

J'ai à nouveau essayé de faire communiquer le client et le serveur UDP entre les deux machines. Mais à nouveau la connexion a été impossible. Le par-feu de l'université empêche l'utilisation des différents ports non affectés des machines, et les personnes disposant des autorisations ne se trouvent pas sur place. Les demandes d'accès ont été faites mais les ports n'ont pas pu être débloqués avant mon départ.

7.3 Lecture de fichier .csv

Il est également possible de réaliser la télé-opération en mode hors-ligne, c'est-à-dire d'enregistrer des mouvements, puis de traiter le fichier ainsi enregistré. Pour cela, il faut pouvoir faire la lecture d'un fichier d'extension .csv. Dans notre cas, il faut pouvoir utiliser le même programme que celui qui est utilisé pour la télé-opération en temps réel. J'ai donc créé une fonction en c++ qui prend en entrée un entier et un fichier .csv et qui permet de lire la ligne correspondant à cet entier. En effet, chaque ligne du fichier .csv correspond à une frame enregistrée, et l'application traite théoriquement chaque frame en temps réel. Il est ainsi possible d'émuler le temps qui passe.

7.4 Comparaison XSens - *Kinect*

7.4.1 Comparaison à l'usage

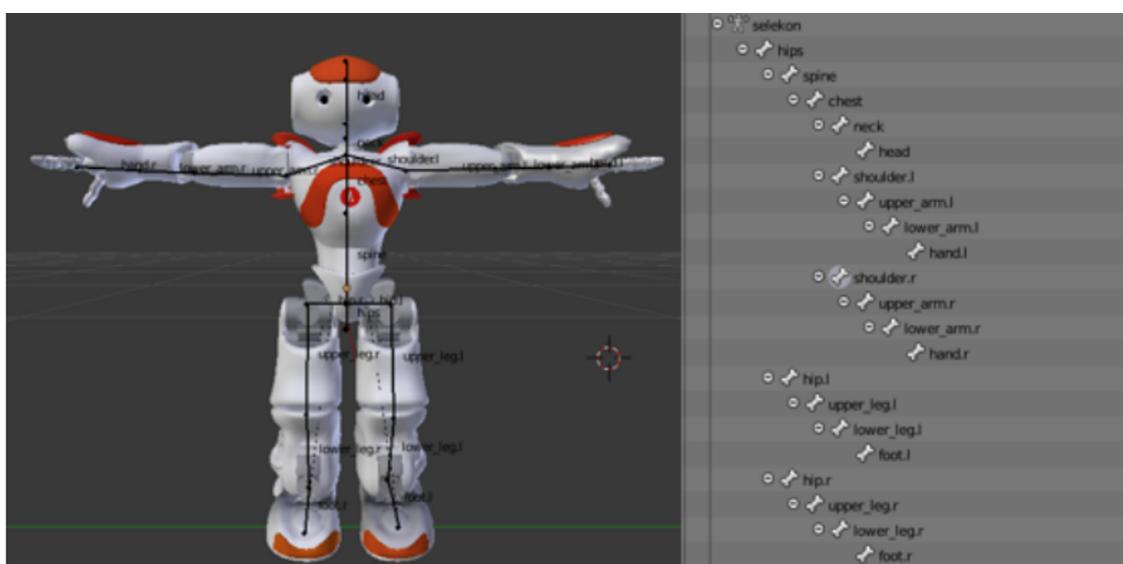
Les technologies Xsens et *Kinect* ont chacune leurs avantages et leurs défauts. A l'usage de ces deux technologies, j'ai pu réaliser un tableau comparatif.

Xsens	Kinect
Très bonne précision et pas de perte de signal	Bonne précision mais perte du signal relativement fréquente
Système très intrusif car nécessite de placer les capteurs sur l'utilisateur	Système non intrusif
Système long à mettre en place à cause de l'équipement de l'utilisateur	Système rapide à mettre en place puisqu'il suffit de placer la caméra Kinect
Système indépendant de l'environnement	Système très dépendant de la couleur et de la texture du fond
Système indépendant des vêtements de l'utilisateur	Système non compatible avec des vêtements amples ou des jupes
Système permettant beaucoup de déplacements grâce à sa grande portée	Système nécessitant d'être bien en face et à une certaine distance (environ 2,5 mètres)
Système utilisable en extérieur	Système très difficilement utilisable en extérieur
Système long à mettre en place et à calibrer	Système rapide à mettre en place et à calibrer

7.4.2 Protocole avec Unity

Pour obtenir d'autres différences, le protocole a été rédigé, mais il n'a malheureusement pas pu être mis en place car il a été suggéré trop tardivement.

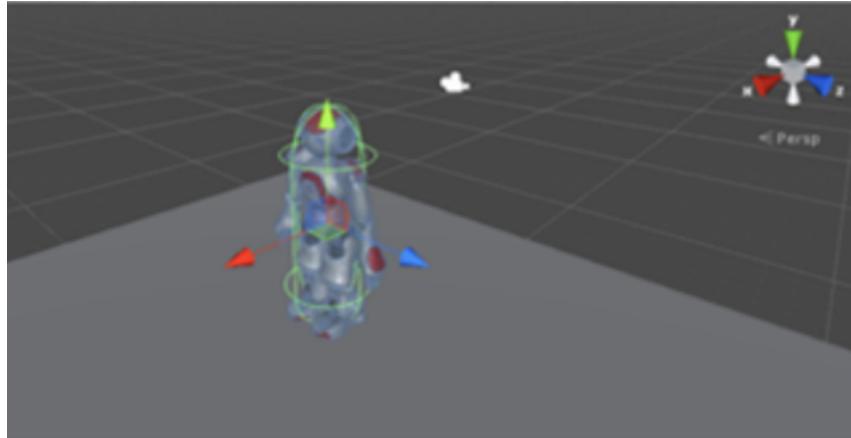
L'idée est de reprendre le travail réalisé par Antenor HERCULE, Farid DEKICHE, Krimo HAMADI, Wafa AYAD et Yanis BOUAZZA. En effet, ils ont réalisé un robot NAO virtuel sous Blender pour pouvoir l'utiliser dans Unity.



Modèle du robot NAO réalisé sous Blender par d'autres stagiaires

Les programmes réalisés par Inaf AJILI permettent d'obtenir cinq fichiers d'extension .csv, qui permettent de déterminer quel mouvement a été détecté.

Sous Unity, il est possible de réaliser un bouton qui va lire les cinq fichiers dans le dossier dont on aura donnée le chemin. Cela permettra de déterminer quel mouvement le robot doit exécuter. Il faut également créer cinq fonctions, une par mouvement que le robot devra effectuer. Lorsque le mouvement à faire sera reconnu, le robot pourra l'exécuter.



Robot NAO virtuel sous Unity

Pour pouvoir comparer réellement XSens et *Kinect*, il est nécessaire de réaliser les enregistrements simultanément. Il faut donc équiper l'utilisateur d'XSens, réaliser le calibrage selon l'axe des X qui sera aussi celui de la *Kinect*, puis démarrer les deux enregistrements simultanément. On pourrait ainsi constater s'il y a un des systèmes qui permet une meilleure reconnaissance que l'autre.

Enfin, cette application pourra être prolongée pour la reconnaissance de geste avec les émotions. En effet, d'autres caractéristiques sont prises en compte pour reconnaître les émotions en plus de celles utilisées pour les gestes. Il est donc possible que les différences s'accentuent ou qu'au contraire elles diminuent.

8 Conclusion et perspectives

Durant ces six mois, j'ai eu l'occasion de participer à divers projets que j'ai pu aider à faire progresser.

J'ai tout d'abord pu faire fonctionner et utiliser la technologie XSens. A présent, il est possible d'utiliser XSens simplement et d'en exploiter les données grâce aux documents que j'ai trouvés et regroupés. A partir de là, j'ai pu mettre en place un protocole scientifique pour collecter une base de données, puis la contrôler avec des tests mathématiques et en vérifiant la concordance de notre travail avec le monde réel. Enfin, la base de données a pu être exploitée pour faire de la reconnaissance de mouvements avec le modèle HMM. Les programmes développés permettent à présent de prendre en entrée une base de données acquise grâce à la Kinect aussi bien que par XSens.

J'ai par ailleurs commencé à travailler sur l'intégration d'XSens au logiciel de téléopération de NAO déjà existant. Même si mon travail n'a pas pu aboutir, j'ai effectué des recherches sur les méthodes à mettre en place et des tests qui permettront de gagner du temps lorsque quelqu'un d'autre travaillera sur ce sujet.

Le travail fourni permet de nombreuses perspectives. Deux axes de travail complémentaires ont d'ores et déjà été identifiés. Tout d'abord, la reconnaissance de geste se fait actuellement en mode hors ligne, c'est-à-dire que le geste est enregistré avant d'être reconnu. L'objectif est que l'utilisateur puisse faire un mouvement devant la *kinect* ou en étant équipé d'XSens et que le geste soit reconnu. Cela implique notamment d'avoir un moyen de détecter le début et la fin du mouvement. La deuxième chose à faire est de réaliser la réaction du robot NAO en fonction du mouvement et de l'émotion reconnus.

Mon travail a donc permis de faire avancer les recherches des deux doctorantes avec lesquelles j'ai pu travailler, mais aussi de permettre de préparer le terrain pour les recherches au sein du laboratoire IBISC, qui permettront de rendre toujours plus simple et naturelles les interactions homme-machine.

Bibliographie

Références

- [1] Burger Brice, Lerasle Frédéric, Ferrané Isabelle, and Aurélie Clodic. Mutual assistance between speech and vision for human-robot interaction. *Intelligent Robots and Systems*, 2002.
- [2] Michael Buettner, Richa Prasad, Matthai Philipose, and David Wetherall. Recognizing daily activities with rfid-based sensors. In *Proceedings of the 11th international conference on Ubiquitous computing*, pages 51–60. ACM, 2009.
- [3] Guangchun Cheng, Yiwen Wan, Abdullah N Saudagar, Kamesh Namuduri, and Bill P Buckles. Advances in human action recognition : A survey. *arXiv preprint arXiv :1501.05964*, 2015.
- [4] Droeuschel David, Stückler Jörg, and Behnke Sven. Learning to interpret pointing gestures with a time-of-flight camera. *Human-Robot Interaction (HRI)*, 2011.
- [5] Hajar Hiyadi, Fakhr-Eddine Ababsa, El Houssine Bouyakhf, Christophe Montagne, and Fakhita Regragui. Reconnaissance 3d des gestes pour l’interaction naturelle homme robot. *15ème édition des journées francophones des jeunes chercheurs en vision par ordinateur (ORASIS 2015)*, pages elec–proc, 2015.
- [6] Pengyu Hong, Matthew Turk, and Thomas S Huang. Gesture modeling and recognition using finite state machines. In *Automatic face and gesture recognition, 2000. proceedings. fourth ieee international conference on*, pages 410–415. IEEE, 2000.
- [7] Oikonomidis Iason, Kyriazis Nikolaos, and Argyros Antonis. Efficient model-based 3d tracking of hand articulations using kinect. *Proceedings of the 22nd British Machine Vision Conference*, 2011.
- [8] Almetwally Ismail and Mallem Malik. Real-time tele-operation and tele-walking of humanoid robot nao using kinect depth camera. *IEEE International Conference*, 2013.
- [9] Biswas K. and Basu Saurav Kumar. Gesture recognition using microsoft kinect. *Automation, Robotics and Applications*, 2011.
- [10] Adistambha Kevin, Ritz Christian, and Burnett Ian. Motion classification using dynamic time warping. *Multimedia Signal Processing*, 2008.
- [11] Koenemann and Bennewitz J. Whole-body imitation of human motions with a nao humanoid. *IEEE International Conference on Human-Robot Interaction (HRI)*, March 2012.
- [12] Bretzner Lars, Laptev Ivan, and Lindeberg Tony. Hand gesture recognition using multi-scale colour features, hierarchical models and particle filtering. *Automatic Face and Gesture Recognition*, 2002.
- [13] Rabiner Lawrence. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 1989.
- [14] Song Le and Takatsuka Masahiro. Real-time 3d finger pointing for an augmented desk. *Proceedings of the Sixth Australasian conference on User interface*, 2005.

- [15] Qin Ling and Lei Bo. Distributed multiagent for nao robot joint position control based on echo state network. In *Mathematical Problems in Engineering*, 2015.
- [16] Pierobon M., Marcon M., Sarti A., and Tubaro S. 3-d body posture tracking for human action template matching, acoustics. *IEEE International Conference on Acoustics*, 2006.
- [17] Elmezain Mahmoud, Al-Hamadi Ayoub, and Michaelis Bernd. Real-time capable system for hand gesture recognition using hidden markov models in stereo color image sequences. 2008.
- [18] Moavenian Majid and Khorrami Hamid. A qualitative comparison of artificial neural networks and support vector machines in ecg arrhythmias classification. *Expert Systems with Applications*, 2010.
- [19] Holte M.B., Moeslund T.B., and Fihl P. View invariant gesture recognition using the csem swissranger sr-2 camera. *International Journal of Intelligent Systems Technologies and Applications*, 2008.
- [20] Van den Bergh Michael, Carton Daniel, De Nijs Roderick, Mitsou Nikos, Landsiedel Christian, Kuehnlenz Kolja, Wollherr Dirk, Van Gool Luc, and Buss Martin. Real-time 3d hand gesture interaction with a robot for understanding directions from humans. *RO-MAN*, 2011.
- [21] Nathan Miller, Odest Chadwicke Jenkins, Marcelo Kallmann, and Maja J Matarić. Motion capture from inertial sensing for untethered humanoid teleoperation. In *Humanoid Robots, 2004 4th IEEE/RAS International Conference on*, volume 2, pages 547–565. IEEE, 2004.
- [22] Patsadu Orasa, Nukoolkit Chakarida, and Watanapa Bunthit. Human gesture recognition using kinect camera. *Computer Science and Software Engineering (JCSSE)*, 2012.
- [23] Senin Pavel. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu*, 2008.
- [24] Breuer Pia, Eckes Christian, and Müller Stefan. Hand gesture recognition with a novel ir time-of-flight range camera – a pilot study. *International Conference on Computer Vision/Computer Graphics Collaboration Techniques and Applications*, 2007.
- [25] Li Qiong, Meng Qinglin, Cai Jiejin, Yoshino Hiroshi, and Mochida Akashi. Predicting hourly cooling load in the building : A comparison of support vector machine and different artificial neural networks. *Energy Conversion and Management*, 2004.
- [26] Stiefelhagen R., Fügen C., Gieselmann P., Holzapfel H., Nickel K., and Waibel A. Natural human-robot interaction using speech, head pose and gestures. *Intelligent Robots and Systems*, 2004.
- [27] Azad Reza and Davami Fatemeh. A robust and adaptable method for face detection based on color probabilistic estimation technique. *International Journal of Research in Computer Science A Unit of White Globe Publications*, 2014.
- [28] I. Rodriguez, A. Astigarraga, E. Jauregi, T. Ruiz, and E. Lazcano. Humanizing nao robot teleoperation using ros. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 179–186, Nov 2014.

- [29] Igorevich Rustam Rakhimov, Park Pusik, Choi Jongchan, and Min Dugki. Two hand gesture recognition using stereo camera. *International Journal of Computer & Electrical Engineering*, 2013.
- [30] Anzalone Salvatore Maria, Tilmont Elodie, Boucenna Sofiane, Chetouani Mohamed, and Cohen David. Tracking posture and head movements of impaired people during interactions with robots. *International Conference on Image Analysis and Processing*, 2013.
- [31] Muhammed Shah, Hussain Abid, and Rashid Harun-ur. User independent hand gesture recognition by accelerated dtw. *Informatics, Electronics & Vision (ICIEV)*, 2012.
- [32] Rautaray Siddharth Swarup and Agrawal Anupam. A real time hand tracking system for interactive applications. *International journal of computer Applications*, 2011.
- [33] Bernhard Spanlang, Xavi Navarro, Jean-Marie Normand, Sameer Kishore, Rodrigo Pizarro, and Mel Slater. Real time whole body motion mapping for avatars and robots. In *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, VRST '13, pages 175–178, New York, NY, USA, 2013. ACM.
- [34] Eickeler Stefan, Kosmala Andreas, and Rigoll Gerhard. Hidden markov model based continuous online gesture recognition. *Pattern Recognition*, 1998.
- [35] Waldherr Stefan, Romero Roseli, and Thrun Sebastian. A gesture based interface for human-robot interaction. *Autonomous Robots*, 2000.
- [36] Pavlovic Vladimir, Sharma Rajeev, and Huang Thomas. Visual interpretation of hand gestures for human-computer interaction : A review. *IEEE Transactions on pattern analysis and machine intelligence*, 1997.
- [37] Pieczynski Wojciech. Modèles de markov en traitements d’images. *Traitements du signal*, 2003.
- [38] Gu Y., Do H., Ou Y., and Sheng W. Human gesture recognition through a kinect sensor. *International Conference on Robotics and Biomimetics (ROBIO)*, 2012.

Xsens

Vidéo expliquant les différentes mesures à prendre pour pouvoir utiliser XSens :
<https://www.youtube.com/watch?v=0sUAYASCy9g>

Vidéo expliquant comment positionner les différents capteurs XSens :
<https://www.youtube.com/watch?v=6TWrcZ84fR4>

Vidéo expliquant comment calibrer le système XSens :
<https://www.youtube.com/watch?v=q7QG07M78ds>

Manuel d'utilisation de MVN : https://xsens.com/download/usermanual/3DBM/MVN_User_Manual.pdf (attention, mis régulièrement à jour, et difficile à trouver par de simples recherches).

Driver ethzasl_xsens_driver pour ROS : http://wiki.ros.org/xsens_driver

Kinect

Driver pour launch la caméra *Kinect* sous ROS : http://wiki.ros.org/openni_launch

Driver pour détecter et suivre les mouvements sous ROS : http://wiki.ros.org/openni_tracker

Glossaire

XSens Awinda

XSens Awinda est un ensemble de capteur inertiels à répartir à des endroits précis du corps pour pouvoir obtenir un suivi 3D de l'utilisateur.

MVN Analyze

MVN Analyze est le logiciel qui permet de récolter et traiter les données fournies par XSens Awinda.

Fichier .mvn

Les fichiers .mvn sont les fichiers créés par défaut par le logiciel MVN Analyze lors de l'enregistrement des données des capteurs XSens.

Fichier .mvnx

Les fichiers .mvnx sont les fichiers .mvn exportés pour pouvoir être lisibles grâce à Excel.

Fichier .csv

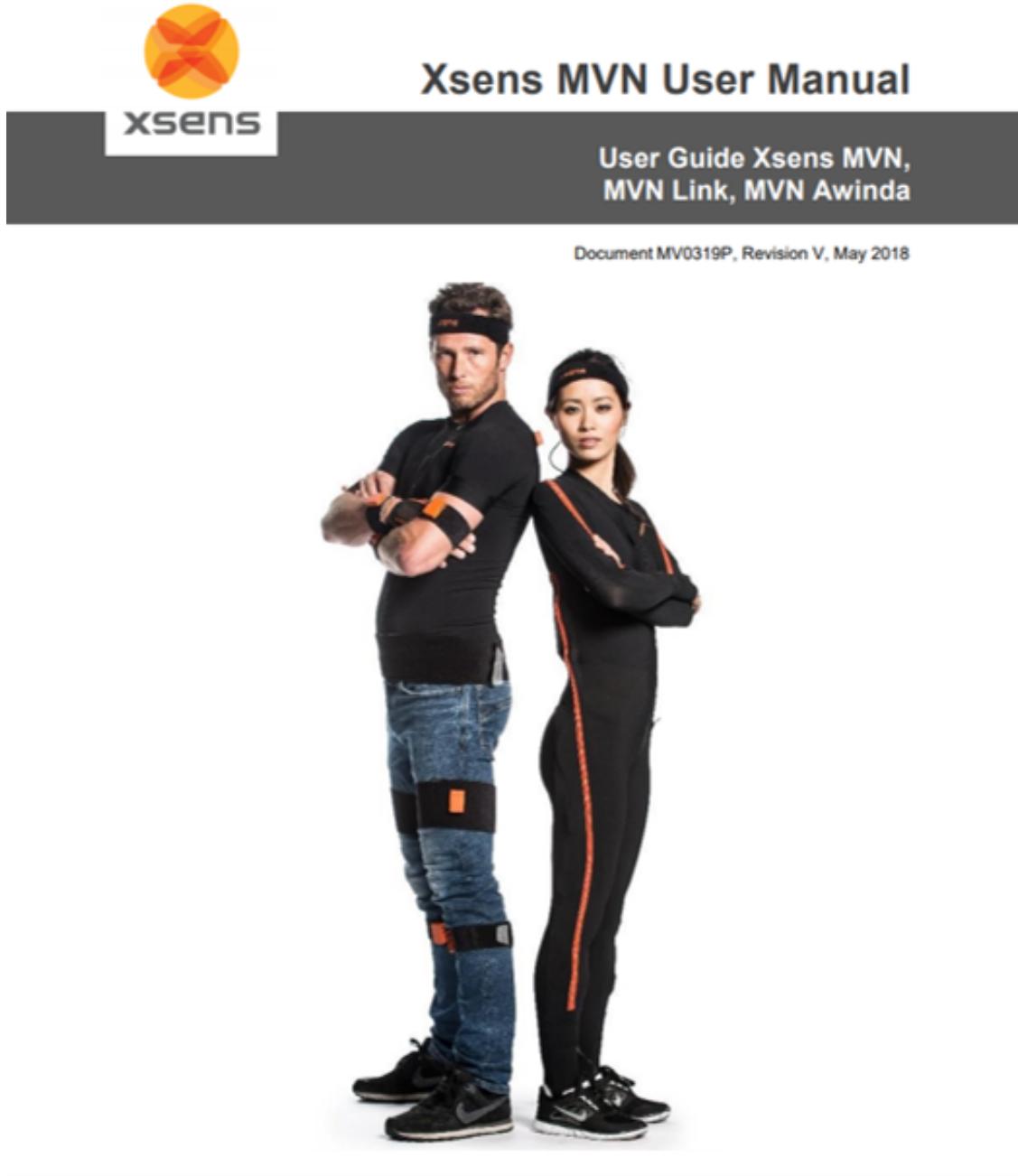
Les fichiers .csv sont des tableaux lisibles à la fois grâce à Excel mais aussi sous forme de texte. C'est le format le plus utilisé pour transporter des données.

ROS

ROS (Robot Operating System) fournit des bibliothèques et des outils pour aider les développeurs de logiciels à créer des applications robotiques sous linux. Il fournit une abstraction matérielle, des pilotes de périphériques, les bibliothèques, les visualiseurs, le passage de messages, la gestion des paquets, et plus encore.

Annexes

Le document le plus utile est le manuel utilisateur, mais c'est aussi le plus dense. En voici un petit extrait avec notamment les documents utiles pour l'analyse et l'interprétation des données. Suit ensuite un document sur le réseau pour pouvoir utiliser XSens sur la reconnaissance de geste en ligne.





Xsens

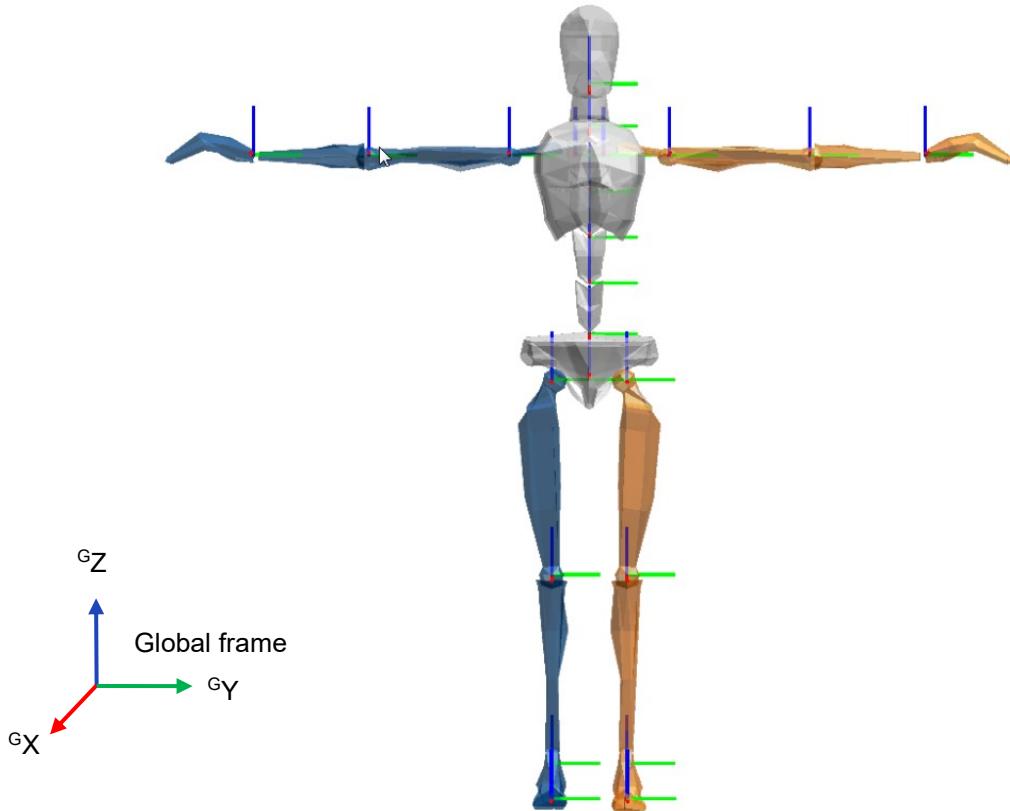


Figure 55: Segment coordinate system at each segment origin, as is used in MVN. Legend: x: red, y: green, z: blue

This section describes the MVNX definition based on version 4 (available since MVN 4.3.7). This version contains some important fixes with respect to the previous version and it is recommended to use this version. Still, version 3 is available and can be chosen in the MVNX exporter preferences. Section 14.4.1 contains more details on the differences between MVNX version 3 and version 4.



Xsens

The entire MVNX structure is as follows:

```
<?xml version="1.0" encoding=" UTF-8"?>
<mvnx version="4">
    <mvn version="..." build="..." />
    <comment></comment>
    <subject label="Suit..." frameRate="..." segmentCount="..." recDate="..." originalFilename="...">
        <comment></comment>
    <segments>
        <segment label="..." id="...">
            <points>
                <point label="...">
                    <pos_b>X Y Z</pos_b>
                </point>
            etc.
            </points>
        </segment>
    etc.
    </segments>
    <sensors>
        <sensor label = "..." />
    etc.
    </sensors>
    <joints>
        <joint label="...">
            <connector1>...</connector1>
            <connector2>...</connector2>
        </joint>
    etc.
    </joints>
    <frames segmentCount= "..." sensorCount= "..." jointCount= "...">
        <frame time= "..." index= "..." type ="normal">
            <orientation>GBqseg1 GBqseg2 ...etc... GBqseg23</orientation>
            <position>Gposseg1 Gposseg1 ...etc... Gposseg23</position>
            <velocity>Gvseg1 Gvseg2 ...etc... Gvseg23</velocity>
            <acceleration>Gaseg1 Gaseg2 ...etc... Gaseg23</acceleration>
            <angularVelocity>Gwseg1 Gwseg2 ...etc... Gwseg23</ angularVelocity >
            <angularAcceleration>Gawseg1 Gawseg2 ...etc... Gawseg23</angularAcceleration>
            <sensorMagneticField>Smsen1 Smsen2 ...etc... Smsen17</sensorMagneticField>
            <sensorOrientation>GSqsen1 GSqsen2 ...etc... GSqsen17</sensorOrientation>
            <jointAngle>jjnt1 jjnt2 ...etc... jjnt22</jointAngle>
            <jointAngleXYZ>jjnt1 Jjnt2 ...etc... Jjnt22</jointAngleXYZ>
            <centerOfMass>GCseg1 GCseg2 ...etc... GCseg23</centerOfMass>
            <marker>"name" </ marker >
        </frame>
    etc.
    </frames>
    </subject>
    <securityCode code= "..." />
</mvnx>
```



XSENS

The MVNX file starts with the XML version number and a reference to the DTD (Document Type Definition). Then the root element “mvnx” with a reference to the XSD (XML Schema Definition) and the MVNX version This is followed by a field “mvn”, containing the MVN Analyze/Animate version and build details with which this MVNX file was produced, followed by the comments that were added to the original recording:

```
<?xml version="1.0" encoding=" UTF-8"?>
<!DOCTYPE mvnx SYSTEM "http://www.xsens.com/mvn/mvnx/schema.dtd">
<mvnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.xsens.com/mvn/mvnx" xsi:schemaLocation="http://www.xsens.com/mvn/mvnx
      http://www.xsens.com/mvn/mvnx/schema.xsd" version="4">
  <mvn version="..." build="..."/>
  <comment></comment>
```

The file continues with session information, including the suit label (= name of the MVN System), sample frequency used, number of segments calculated (indicating full or half body); the date that the recording took place, and the filename of the original recording on which the MVNX file is based. When creating a recorded file, a user can insert a comment. This is available in the exported MVNX: (this field usually has the same contents as the comment field after the <mvn> tag, see above)

```
<subject label=" Suit 00130268" frameRate="100 segmentCount="23" recDate="day month
year" originalFilename="DIRECTORY\FILENAME.mvn">
  <comment></comment>
```

The next section contains mesh scale and labeling information for the segments. The mesh scale data is used for scaling of the visualization of the character in MVN Analyze/Animate.

The section <segments> defines all positions (pos_b) of connecting joints (prefixed with a “j”) and anatomical landmarks (prefixed with a “p”) with respect to origin of that segment (in body fame B). The body frame (B) of this list of points is defined by Figure 55.

Within each segment is:

```
<segments>
  <segment label="RightUpperLeg" id="16">
    <points>
      <point label="...">
        <pos_b>X Y Z</pos_b>
      </point>
    etc.
    </points>
  </segment>
```

The sensor data is simply a list of the names of segments measured with the MT's:

```
<sensors>
  <sensor label ="Pelvis"/>
  <sensor label ="Head"/>
  <sensor label ="RightShoulder"/>
etc.
</sensors>
```



XSENS

The “joints” section is a list of the names of joints contained and the segments and connections a given joint:

```
<joints>
<joint label=" jLeftHip">
<connector1>Pelvis/jLeftHip </connector1>
<connector2>LeftUpperLeg/jLeftHip </connector2>
</joint>
etc.
</joints>
```

The frames section is opened with the segment, sensor and joint count. The data of all parameters are contained within this section, including the calibration pose data.

```
<frames segmentCount= "..." sensorCount= "..." jointCount= "...>
<frame ... >
</frame>
Etc.
</frames>
```

The frame type “identity” denotes the nullpose or identity pose. All segment orientations in this pose are aligned with the global coordinates and have unit quaternion.

```
<frame time="0" type="identity">
    <orientation>...<orientation/>
    <position>...<position/>
</frame>
```

The frame type “tpose” describes the positions and orientations of all segments in the T-pose. The positions and orientations of certain segments deviates slightly from the identity pose.

```
<frame time="0" type="tpose">
    <orientation>...<orientation/>
    <position>...<position/>
</frame>
```

The frame type “tpose-isb” describes the positions and orientations of all segments in the T-pose, but using the MVN anatomical frame for the body segments (Figure 60). The MVN anatomical frame is used to calculate the joint angles.

```
<frame time="0" type="tpose-isb">
    <orientation>...<orientation/>
    <position>...<position/>
</frame>
```

Following this, the frames are split into *time*; each frame increment is equivalent to ([frame no. /update rate]*1000 [ms]). So at 120Hz, frame 1 is denoted by ($[1/120]*1000 = 8.3$ [ms]). Frames measured during a normal session are denoted with type = “normal”.

The *index* attribute still contains the original frame number.

The optional “tc” and “ms” attributes contain the TimeCode values, i.e. the absolute real time at which that specific frame was recorded (often used to synchronize and combine this recording with other data that was gathered at the same time).



XSENS

The selection in the preferences menu determines the MVNX contents. Orientation and Position are mandatory, all others are optional and may depend on the license.

```
<frame time="..." index="frame number" tc="15:57:47:02" ms="1418227067039" type="normal">
<orientation>GBqseg1 GBqseg2 ...etc... GBqseg23</orientation>
<position>Gposseg1 Gposseg2 ...etc... Gposseg23</position>
<velocity>Gvseg1 Gvseg2 ...etc... Gvseg23</velocity>
<acceleration>Gaseg1 Gaseg2 ...etc... Gaseg23</acceleration>
<angularVelocity>Gwseg1 Gwseg2 ...etc... Gwseg23</angularVelocity>
<angularAcceleration>Gawseg1 Gawseg2 ...etc... Gawseg23</angularAcceleration>
<sensorMagneticField>Smsen1 Smsen2 ...etc... Smsen17</sensorMagneticField>
<sensorOrientation>GSqsen1 GSqsen2 ...etc... GSqsen17</sensorOrientation>
<jointAngle>jjnt1 jjnt2 ...etc... jjnt22</jointAngle>
<jointAngleXZY>jjntx1 jjntx2 ...etc... jjntx22</jointAngleXZY>
<centerOfMass>GCseg1 GCseg2 ...etc... GCseg23</centerOfMass>
<marker>"name" </ marker name>
</frame>
```

For the elements represented below with a subscript "seg", the set will contain the number of segments (segmentCount) times the number of columns of data. For a full body, this is 23 segments.

Subscript "sen" contains the number of MT's (sensorCount) times the number of columns of data. For a full body, this is 17.

Subscript "jnt" contains the number of joints (jointCount) times the number of columns of data. For a full body, this is 22 joints.

^{GB} q _{seg}	1x4 quaternion vector (q ₀ , q ₁ , q ₂ , q ₃) describing the orientation of the segment with respect to the global frame.
^G pos _{seg}	1x3 position vector (x, y, z) of the origin of the segment in the global frame in [m].
^G v _{seg}	1x3 velocity vector (x, y, z) of the origin of the segment in the global frame in [m/s].
^G a _{seg}	1x3 acceleration vector (x, y, z) of the origin of the segment in the global frame in [m/s ²].
^G w _{seg}	1x3 angular velocity vector (x, y, z) of the segment in the global frame in [rad/s].
^G a _{wseg}	1x3 angular acceleration vector (x, y, z) of the origin of the segment in the global frame in [rad/s ²].
^G a _{sen}	1x3 sensor free acceleration vector (x, y, z) of the sensor in [m/s ²].
^S m _{sen}	1x3 sensor magnetic field vector (x, y, z) of the sensor in [a.u.].
^{GS} q _{sen}	1x4 sensor orientation quaternion (q ₀ , q ₁ , q ₂ , q ₃) of the sensor in the global frame in.
j _{jnt}	1x3 Euler representation of the joint angle vector (x, y, z) in [deg], calculated using the Euler sequence ZXY using the ISB based coordinate system.
j _{jntx}	1x3 Euler representation of the joint angle vector (x, y, z) in [deg], calculated using the Euler sequence XZY using the ISB based coordinate system. <i>Note: The joint angle using Euler sequence XZY is calculated and exported for all joints, but commonly only used for the shoulder joints, and it may depend on the movement of the shoulder if it is appropriate to use.</i>
^G C _{seg}	1x3 position of the body Center of Mass (x,y,z) in the global frame in [m].



XSENS

The numbering is presented in the table below.

Number	Segment Label	Tracker	Joint
1	Pelvis	Pelvis	jL5S1
2	L5	T8	jL4L3
3	L3	Head	jL1T12
4	T12	RightShoulder	jT9T8
5	T8	RightUpperArm	jT1C7
6	Neck	RightForeArm	jC1Head
7	Head	RightHand	jRightC7Shoulder
8	Right Shoulder	LeftShoulder	jRightShoulder
9	Right Upper Arm	LeftUpperArm	jRightElbow
10	Right Forearm	LeftForeArm	jRightWrist
11	Right Hand	LeftHand	jLeftC7Shoulder
12	Left Shoulder	RightUpperLeg	jLeftShoulder
13	Left Upper Arm	RightLowerLeg	jLeftElbow
14	Left Forearm	RightFoot	jLeftWrist
15	Left Hand	LeftUpperLeg	jRightHip
16	Right Upper Leg	LeftLowerLeg	jRightKnee
17	Right Lower Leg	LeftFoot	jRightAnkle
18	Right Foot		jRightBallFoot
19	Right Toe		jLeftHip
20	Left Upper Leg		jLeftKnee
21	Left Lower Leg		jLeftAnkle
22	Left Foot		jLeftBallFoot
23	Left Toe		



XSENS

14.4.1 MVNX backwards compatibility

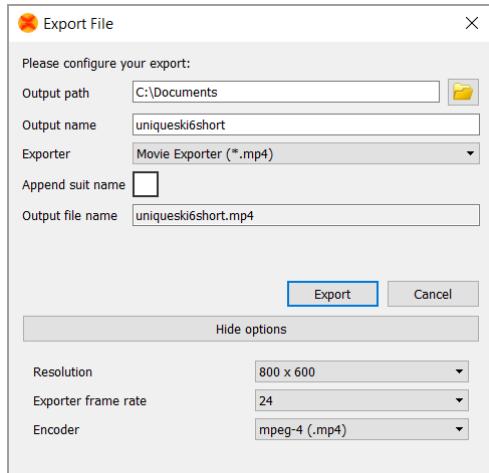
Until MVN 4.3.7, MVNX (version 3) contained some inconsistencies related to the coordinate frames, which have been resolved (version 4). For backwards compatibility, it is still possible to import and export version 3 of MVNX.

The changes of version 4 with respect to version 3 are:

- The list of points at the beginning of the MVNX files is now correctly denoted as pos_b instead of pos_s.
- The segment orientations ${}^{GB}q$ give the relation between global frame (G) and body frame (B). In version 3, this body frame is defined in Figure 55 and consistently used in the MVNX. In version 3, the body frames are defined by the MVN anatomical pose (Figure 60).
- The initial poses used by the MVNX are now written as “identity” denoting the null-pose, “tpose” denoting the T-pose, and “tpose-isb” denoting the T-pose using the MVN anatomical coordinate frame. In version 3, the initial poses were “tpose” denoting the T-pose using MVN anatomical coordinate frame, and “npose” denoting the relation between the body frames of the T-pose and MVN anatomical pose.

14.5 Export Movie

Exporting movie data enables the user to export the 3D viewport of MVN Analyze/Animate to either .m4v or .avi. this facilitates presenting MVN information to audiences without the need for installation of MVN Analyze/Animate. Select the export format, resolution and export frame in >Options >Preferences >Exporters >Movie Exporter.





XSENS

15 Features of MVN Analyze/Animate

A range of additional features become available when using another license for MVN Animate Pro. These are summarized in Table 6.

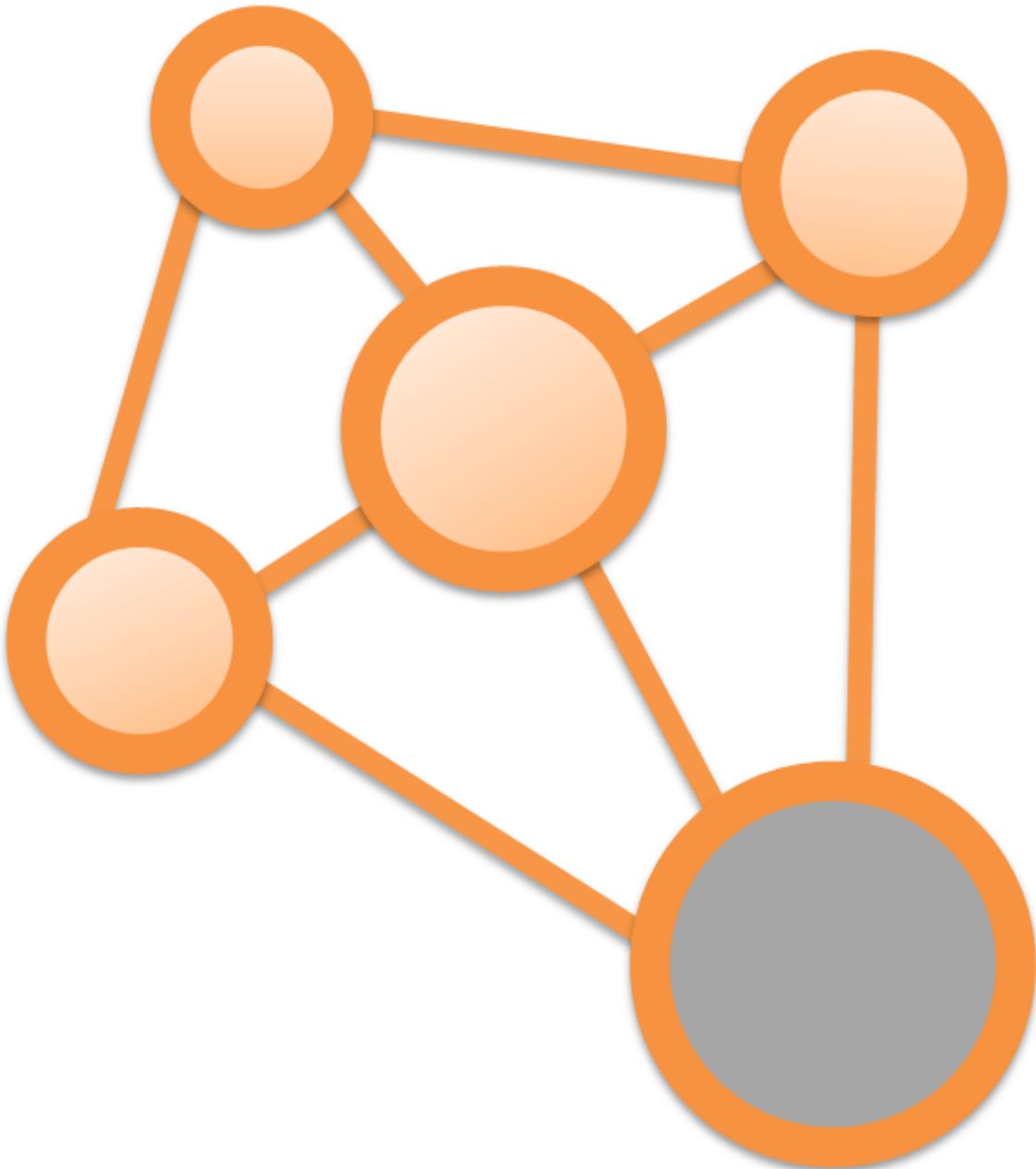
Table 6: Overview of features for each possible MVN Analyze/Animate license

Features	MVN Animate	MVN Animate Pro	MVN Analyze
Multi-person	✓	✓	✓
Network Streamer (plug-ins)	✗	✓	✓
Extended Network Streamer	✗	✗	✓
Limited configurations	✓	All	All
Limited scenarios	✓	All	All
Reference camera support	✗	✓	✓
Export as movie	✗	✓	✓
Time code & remote control	✗	✓	✓
Graphs	✗	✓	✓
Real-time graphs	✗	✗	✓
Extended MVNX	✗	✗	✓
Third party synchronization	✗	✗	✓
Display biomechanical model	✗	✗	✓

MVN Analyze/Animate real-time network streaming

Protocol Specification

Document MV0305P, Revision L, April 2018



Xsens Technologies B.V.

Pantheon 6a
P.O. Box 559
7500 AN Enschede
The Netherlands

phone +31 (0)88 973 67 00
fax +31 (0)88 973 67 01
e-mail info@xsens.com
internet www.xsens.com

Xsens North America, Inc.

10557 Jefferson Blvd,
Suite C
CA-90232 Culver City
USA

phone 310-481-1800
fax 310-416-9044
e-mail info@xsens.com
internet www.xsens.com



Revisions

Revision	Date	By	Changes
E	February 2012	DOS	Updated for release MVN Analyze/Animate 3.3
F	June 2013	AOD	Updated for release MVN Analyze/Animate 3.5
G	December 2013	CMO	Updated for release MVN Analyze/Animate 3.5.2
H	October 2014	JMU	Updated for release MVN Analyze/Animate 4.0 (new HW + support for TCP protocol)
I	November 2014	PVR	Indicate more clearly which data types are used by MotionBuilder, Maya and Unity3D
J	February 2015	JMU	Updated for MVN Analyze/Animate 4.1: Additional datagrams for expanded network streaming defined
K	November 2017	HBE	Updated naming for MVN 2018
L	April 2018	EJO	Updated documentation for scale information

© 2005-2018, Xsens Technologies B.V. All rights reserved. Information in this document is subject to change without notice. Xsens, MVN, MotionGrid, MTi, MTi-G, MTx, MTw, Awinda and KiC are registered trademarks or trademarks of Xsens Technologies B.V. and/or its parent, subsidiaries and/or affiliates in The Netherlands, the USA and/or other countries. All other trademarks are the property of their respective owners.



Table of Contents

1	INTRODUCTION	1
1.1	PERCEIVED USAGE	1
1.1.1	<i>Usage in real-time previsualization and simulation VR setups</i>	1
1.1.2	<i>Network Streamer and Network monitor.....</i>	1
1.1.3	<i>Usage in multi-person or other complex motion capture setups</i>	2
2	TRANSPORT MEDIUM	3
2.1	NETWORK ENVIRONMENT.....	3
2.2	NETWORK PROTOCOL	3
2.3	DEFAULT PORT	3
2.4	DATAGRAM	3
2.4.1	<i>Header</i>	3
2.5	POSE DATA	6
2.5.1	<i>Segment data Euler (type 01)</i>	6
2.5.2	<i>Segment data quaternion (type 02).....</i>	7
2.5.3	<i>Point position data (type 03)</i>	7
2.5.4	<i>MotionGrid Tag data (type 04)</i>	7
2.5.5	<i>Segment data Unity3D (type 05).....</i>	7
2.5.6	<i>Position</i>	8
2.5.7	<i>Rotation (Euler)</i>	8
2.5.8	<i>Rotation (Quaternion).....</i>	8
2.5.9	<i>Segment ID</i>	8
2.5.10	<i>Point ID</i>	8
2.5.11	<i>Float and integer values over the network</i>	8
2.5.12	<i>String values over the network</i>	8
2.6	CHARACTER INFORMATION	8
2.6.1	<i>Scale information (type 10)</i>	8
2.6.2	<i>Prop information (type 11)</i>	9
2.6.3	<i>Meta data (type 12).....</i>	9
2.6.4	<i>Scale information (type 13)</i>	9
2.7	ADDITIONAL INFORMATION	9
2.7.1	<i>Joint Angles (type 20).....</i>	10
2.7.2	<i>Linear Segment Kinematics (type 21).....</i>	10
2.7.3	<i>Angular Segment Kinematics (type 22)</i>	10
2.7.4	<i>Motion Tracker Kinematics (type 23).....</i>	11
2.7.5	<i>Center of Mass (type 24)</i>	11
2.7.6	<i>Time Code (type 25).....</i>	11
3	DATA TYPES	12
3.1	SEGMENT IDs.....	12



1 Introduction

MVN Analyze/Animate, developed by Xsens, is a tool to capture and compute the 6DOF motion data of an inertial sensor-driven system. It allows the export of the data to third party applications such as Motion Builder, making the data available to drive rigged characters in e.g. animations. The data transfer to other applications is primarily file based when using MVN Analyze/Animate. With the XME API (SDK) there are many other options.

In many situations it is attractive to keep the ease of use of MVN Analyze/Animate, while receiving the motion capture data in real-time in another application, even on another PC possibly physically remote from the MVN system.

This document defines a network protocol specification for this purpose. It describes the transport medium, the given data and the datagrams to be sent and received over the network, as well as the control sequences the server and clients will use to communicate states and requests during the sessions. The network communication is mainly required to be fast/real-time, other quality criteria are secondary.

This document describes MVN Analyze/Animate Real-time Network Streaming. The streaming feature enables computers that run MVN Analyze/Animate to stream the captured data over a network to other client computers.

1.1 Perceived Usage

1.1.1 Usage in real-time previsualization and simulation VR setups

Many software packages (e.g. MotionBuilder) and experimental VR rigs use single computers to do specific processing and hardware interfacing tasks, such as driving motion platforms, real-time rendering to a screen, or interfacing with a motion capture device. In this scenario, a PC set up with MVN Analyze/Animate could service one (or more) motion captured persons. This requires immediate, regularly timed delivery of state (pose) packets. The UDP protocol is most suitable for this task because it delivers packets without congestion control and dropped packet checking. MVN Analyze/Animate real-time network streaming protocol is based on UDP and is specified in this document.

To support scenarios like this for usage with 3rd party tools as a client application, Xsens has developed several plug-ins. MVN Analyze/Animate plug-ins are available for **Autodesk Motion Builder**, **Autodesk Maya** and **Unity3D**. These tools use protocols specified in this document to receive motion capture data in real-time.

The client side plug-ins for MotionBuilder and Maya can be requested and purchased separately at Xsens.

The Unity3D plug-in is available for free at: <https://www.assetstore.unity3d.com/en/#!/content/11338>
(Version: [1.0](#) (Apr 25, 2014), Size: 1.6 MB, this requires Unity 4.0.1 or higher)

1.1.2 Network Streamer and Network monitor

To send motions from MVN Analyze/Animate, go to Options > Preferences > Miscellaneous > Network Streamer and choose the desired protocol. The motion can also be received by MVN Analyze/Animate go to File > open Network Monitor. The Network Monitor will also show the local axis for each segment. Also note that the red triangle in the origin is representing the x-axis.



1.1.3 Usage in multi-person or other complex motion capture setups

In roll-your-own motion capture setups, often additional data is captured. An example could be medical data, or data gloves. Another setup might capture multiple subjects at once. The TCP protocol would be most suitable for this task as this protocol guarantees that the data stream is completely sent, potentially at the expense of near real-time delivery. However UDP also suffices in a well-designed network setup as there will be nearly no, or very little, packet loss.

Advantages for motion capture setup builders include:

- Not necessary to interface with XME API (SDK).
- Processing CPU time required for inertial motion capture is done on a separate PC, freeing up resources for other processing;
- Calibration and real-time pre-viewing (e.g. for assessment of motion capture quality) can be done on the processing PC using MVN Analyze/Animate itself.



2 Transport Medium

2.1 Network Environment

The network environment will be assumed to be a local 100 Mbit Ethernet network, larger network topologies are not considered and can be covered by file transfer of the already given file export functionality or later extensions to the network protocol. Thus, few packet loss or data corruption during transfer is to be expected, as well as constant connectivity.

2.2 Network Protocol

Network communication uses a protocol stack, thus the streaming protocol will be implemented on top of a given set of protocols already available for the network clients. In this case, the layers to build upon are IP and UDP (or TCP, which is also supported). **IP** (Internet Protocol, RFC 791) is the network layer protocol used in Ethernet networks and defines the source and destination of the packets within the network. Upon this, **UDP** (User Datagram Protocol, RFC 768) is used to encapsulate the data. The **UDP** Protocol is unidirectional, and contrary to **TCP** (Transmission Control Protocol, RFC 793) it is stateless and does not require the receiver to answer incoming packets. This allows greater speed.

2.3 Default Port

The default Port to be used on the network is 9763. This Port is derived from the XME API (9=X, M=6, E=3). MVN Analyze/Animate server will default to this Port.

It is of course possible to define an arbitrary Port if needed.

2.4 Datagram

The motion capture data is sampled and sent at regular time intervals for which the length depends upon the configuration of MVN Analyze/Animate. Common sampling rates lie between 60 and 240 Hertz. The update rate of the real-time network stream can be modified separately. The data content in the datagram is defined by the specific protocol set, but basically, the positions and rotation of all segments of the body at a sampling instance are sent away as one or more UDP datagrams.

Each datagram starts with a 24-byte header followed by a variable number of bytes for each body segment, depending on the selected data protocol. All data is sent in ‘network byte order’, which corresponds to big-endian notation.

Framed text indicates items that are sent as part of the datagram.

2.4.1 Header

The header contains the type of the data and some identification information, so the receiving end can apply it to the right target.

Datagram header

6 bytes ID String
4 bytes sample counter
1 byte datagram counter
1 byte number of items
4 bytes time code
1 byte character ID
7 bytes reserved for future use



2.4.1.1 ID String

The so-called ID String is an ASCII string which consists of 6 characters (not terminated by a null character). It serves to unambiguously identify the UDP datagram as containing motion data of the format according to this specification. Since the values in the string are characters, this string is not converted to a big-endian notation, but the first byte is simply the first character, etc.

These are the ASCII and hexadecimal byte values of the ID String:

ASCII	M	X	T	P	0	1
Hex	4D	58	54	50	30	31

M: M for MVN

X: X for Xsens

T: T for Transfer

P: P for Protocol

##: Message type. The first digit determines what kind of packet this is and the second digit determines the format of the data in the packet

Message type	Description
01	Pose data (Euler) ← MotionBuilder +Maya <ul style="list-style-type: none">Absolute position and orientation (Euler) of segmentsY-Up, right-handedThis type is used by the Motion Builder + Maya plug-in<i>Supported by MVN Analyze/Animate network monitor</i>
02	Pose data (Quaternion) ← MVN Analyze/Animate Network Monitor <ul style="list-style-type: none">Absolute position and orientation (Quaternion) of segmentsDefault mode Z-Up, right-handed or Y-Up<i>Supported by MVN Analyze/Animate network monitor</i>
03	Pose data (Positions only, MVN Optical marker set 1) <ul style="list-style-type: none">Positions of selected defined points (simulating optical markers), typically 38-46 points. Multiple data sets are available.This datagram is used by the Motion Builder plug-in v1.0.Partially supported by MVN Analyze/Animate network monitor. MVN Analyze/Animate has a limited ability to re-integrate these marker positions into a character. The segment orientations will not be updated. Therefore, when <u>only this datagram</u> is received, the resulting character can appear incorrect.
04	Deprecated: MotionGrid Tag data
05	Pose data (Unity3D) <ul style="list-style-type: none">Relative position and orientation (Quaternion) of segmentsUses alternative segment orderLeft-handed for Unity3D protocol<i>Supported by MVN Analyze/Animate network monitor</i>
10	Deprecated, use 13: Character information → scale information
11	Deprecated, use 13: Character information → prop information
12	Character information → meta data <ul style="list-style-type: none">name of the characterMVN character ID (BodyPack or Awinda Station ID)<< more can be added later >><i>Supported by MVN Analyze/Animate network monitor</i>



Message type	Description
13	Character information → scaling information, including prop and null-pose <i>Supported by MVN Analyze/Animate network monitor</i>
20	Joint Angle data <ul style="list-style-type: none">• Joint definition and angles• NOT supported by MVN Analyze/Animate network monitor.
21	Linear Segment Kinematics <ul style="list-style-type: none">• Absolute segment position, velocity and acceleration• Partially supported by MVN Analyze/Animate network monitor. <i>MVN Analyze/Animate has a limited ability to re-integrate this data into a character. The segment orientations will not be updated. Therefore, when <u>only this datagram</u> is received, the resulting character can appear incorrect.</i>
22	Angular Segment Kinematics <ul style="list-style-type: none">• Absolute segment orientation, angular velocity and angular acceleration• Partially supported by MVN Analyze/Animate network monitor. <i>MVN Analyze/Animate has a limited ability to re-integrate this data into a character. The segment positions will not be updated. Therefore, when <u>only this datagram</u> is received, the resulting character can appear incorrect.</i>
23	Motion Tracker Kinematics <ul style="list-style-type: none">• Absolute sensor orientation and free acceleration• Sensor-local acceleration, angular velocity and magnetic field• NOT supported by MVN Analyze/Animate network monitor.
24	Center of Mass <ul style="list-style-type: none">• Absolute position of center of mass• NOT supported by MVN Analyze/Animate network monitor.
25	Time Code <ul style="list-style-type: none">• Time code string• NOT supported by MVN Analyze/Animate network monitor.

Please note that the message type is sent as a string, not as a number, so message type “03” is sent as hex code 0x30 0x33, not as 0x00 0x03.

2.4.1.2 Sample Counter

The sample counter is a 32-bit unsigned integer value which is incremented by one, each time a new set of motion tracker data is sampled and sent away. Note that the sample counter is not to be interpreted as a time code, since the sender may skip frames.

2.4.1.3 Datagram Counter

The size of a UDP datagram is usually limited by the MTU (maximum transmission unit, approx. 1500 bytes) of the underlying Ethernet network. In nearly all cases the entire motion data that was collected at one sampling instance will fit into a single UDP datagram. However, if the amount of motion data becomes too large then the data is split up into several datagrams.

If motion data is split up into several datagrams then the datagrams receive index numbers starting at zero. The datagram counter is a 7-bit unsigned integer value which stores this index number. The most significant bit of the datagram counter byte is used to signal that this datagram is the last one belonging to that sampling instance. For example, if motion data is split up into three datagrams then their datagram counters will have the values 0, 1 and 0x82 (hexadecimal). If all data fits into one UDP datagram (the usual case) then the datagram counter will be equal to 0x80 (hexadecimal).

The sample counter mentioned above can be used to identify which datagrams belong to the same sampling instance because they must all carry the same sample counter value but different datagram



counters. This also means that the combination of sample counter and datagram counter values is unique for each UDP datagram containing (part of the) motion data.

NOTE: For practical purposes this will not be an issue with the MVN streaming protocol. If problems are encountered, check your MTU settings.

2.4.1.4 Number of items

The number of items is stored as an 8-bit unsigned integer value. This number indicates the number of segments or points that are contained in the packet. Note that this number is not necessarily equal to the total number of motion trackers that were captured at the sampling instance if the motion capture data was split up into several datagrams. This number may instead be used to verify that the entire UDP datagram has been fully received by calculating the expected size of the datagram and comparing it to the actual size of the datagram.

2.4.1.5 Time code

MVN Analyze/Animate contains a clock which starts running at the start of a recording. The clock measures the elapsed time in milliseconds. Whenever new captured data is sampled the current value of the clock is sampled as well and is stored inside the datagram(s) as a 32-bit unsigned integer value representing a time code.

2.4.1.6 Character ID

MVN Analyze/Animate now supports multiple characters in one viewport. This byte specifies to which character the data belongs. In a single-character setup this value will always be 0. In multi-character cases, they will *usually* be incremental. However, especially during live streaming, one of the characters may disconnect and stop sending data while others will continue, so the receiver should be able to handle this.

Each character will send its own full packet.

2.4.1.7 Reserved bytes for future use

The left-over bytes at the end of the datagram header are reserved for future versions of this protocol.

2.5 Pose data

2.5.1 Segment data Euler (type 01)

This protocol was originally developed and optimized for the MotionBuilder and Maya plug-in.

Information about each segment is sent as follows.

4 bytes segment ID See 2.5.9
4 bytes x–coordinate of segment position
4 bytes y–coordinate of segment position
4 bytes z–coordinate of segment position
4 bytes x rotation –coordinate of segment rotation
4 bytes y rotation –coordinate of segment rotation
4 bytes z rotation –coordinate of segment rotation

Total: 28 bytes per segment

The coordinates use a Y-Up, right-handed coordinate system for Euler protocol.

The number of segments recorded will be sent, followed by the amount of props used, if any. Maximum number of segments is 23, for full body and maximum number of props is 4.



2.5.2 Segment data quaternion (type 02)

This protocol reflects the internal format of MVN Analyze/Animate.

Information about each segment is sent as follows.

```
4 bytes segment ID See 2.5.9
4 bytes x–coordinate of segment position
4 bytes y–coordinate of segment position
4 bytes z–coordinate of segment position
4 bytes q1 rotation – segment rotation quaternion component 1 (re)
4 bytes q2 rotation – segment rotation quaternion component 1 (i)
4 bytes q3 rotation – segment rotation quaternion component 1 (j)
4 bytes q4 rotation – segment rotation quaternion component 1 (k)
```

Total: 32 bytes per segment

The coordinates use a Z-Up, right-handed coordinate system.

The number of segments recorded will be sent, followed by the amount of props used, if any. Maximum number of segments is 23, for full body and maximum number of props is 4.

2.5.3 Point position data (type 03)

Information about each point is sent as follows.

This data type is intended to emulate a Virtual (optical) Marker Set.

```
4 bytes point ID
this is 100x the segment ID + the point ID for a marker
this is the tagId for a tag
4 bytes x–coordinate of point position
4 bytes y–coordinate of point position
4 bytes z–coordinate of point position
```

Total: 16 bytes per point

The coordinates use a Y-Up, right-handed coordinate system.

After the points all MotionGrid tags assigned to the character will be sent, using the same position format as the markers.

2.5.4 MotionGrid Tag data (type 04)

This message has become deprecated.

2.5.5 Segment data Unity3D (type 05)

Information about each segment is sent as follows.

```
4 bytes segment ID See 2.5.9
4 bytes x–coordinate of segment position
4 bytes y–coordinate of segment position
4 bytes z–coordinate of segment position
4 bytes q1 rotation – segment rotation quaternion component 1 (re)
4 bytes q2 rotation – segment rotation quaternion component 1 (i)
4 bytes q3 rotation – segment rotation quaternion component 1 (j)
4 bytes q4 rotation – segment rotation quaternion component 1 (k)
```

Total: 32 bytes per segment.



The pelvis segment uses global positions and rotation, while the other segments only use local rotation and relative positions. Segments follow pelvis position based on the character model hierarchy within Unity3D.

Unity3D mode uses quaternion data, where the coordinates use a Y-Up, left-handed coordinate system.

A total of 23 segments will be sent. Props are not supported.

2.5.6 Position

The position of a captured segment is always stored as a 3D vector composed of three 32-bit float values. The unit is cm.

2.5.7 Rotation (Euler)

The rotation of a captured segment in the Euler representation is always stored as a 3D vector composed of three 32-bit float values. The unit is degrees.

2.5.8 Rotation (Quaternion)

The rotation of a captured segment in the Quaternion representation is always stored as a 4D vector composed of four 32-bit float values. The quaternion is always normalized, but not necessarily positive-definite.

2.5.9 Segment ID

The IDs of the segments are listed in paragraph 3.1. The segment ID is sent as a normal 4-byte integer.

2.5.10 Point ID

Note that since many more options have been added to the streamed data of MVN Analyze/Animate 4.1, the following section contains new information.

The ID of a point depends on the ID of the segment it is attached to and the local ID it has in the segment. These local IDs are documented in the MVN User Manual. The ID is sent as a 4-byte integer, defined as $256 * \text{segment ID} + \text{local point ID}$.

Example:

The Sacrum point on the Pelvis segment has local ID 13, and the Pelvis has ID 1, so the ID of the point is sent as $256 * 1 + 13 = 269$.

2.5.11 Float and integer values over the network

All integer values mentioned above are stored in big-endian byte order inside the UDP datagrams with the function htonl() into the network by MVN Analyze/Animate and ntohs() out in the client. In other words: the most significant byte (MSB) is stored first. This is the same byte order that is used for other Internet protocols, so standard conversion functions should be available on all computer systems.

2.5.12 String values over the network

Strings are utf-8 encoded. They are preceded by the size of the string as a 32-bit signed integer and NOT 0-terminated.

2.6 Character information

2.6.1 Scale information (type 10)

This message has become deprecated. It is superseded by message 13.



2.6.2 Prop information (type 11)

This message has become deprecated. It is superseded by message 13.

2.6.3 Meta data (type 12)

This packet contains some meta-data about the character. This is in a tagged format, each tag is formatted as “tagname:” and each tagline is terminated by a newline. Each value is a string that can be interpreted in its own way.

Defined tags are:

name: contains the name as displayed in MVN Analyze/Animate

xmid: contains the BodyPack/Awinda-station ID as shown in MVN Analyze/Animate

color: contains the color of the character as used in MVN Analyze/Animate, the format is hex RRGGBB

More tags may be added later, so any implementation should be able to skip unknown and unused tags. This packet may contain different tags each time to reduce network load. The order of the tags can vary from packet to packet.

2.6.4 Scale information (type 13)

This packet contains scaling information about the character.

The scaling information is provided as known key points of the character standing in a perfect/ideal Tpose.

The data sent per item for each packet is:

4 bytes: segment count (S)

S times:

- string: name of segment

- 3x4 bytes: x,y,z coordinates of the origin of the segment in global space

4 bytes: point count (P)

P times:

- 2 bytes: segment ID of the point

- 2 bytes: point ID of the point in this segment

- string: name of the point

- 3x4 bytes: x,y,z coordinates of the point in global space

Total: unknown

Typically multiple packets of this type are sent to provide the full scaling information.

The separation is done to allow successful communication on networks with typical packet size limitations.

The first packet in a sequence will contain just the segments and will set the point count to 0.

Following this will be packets with the segment count set to 0 and the point count set to a non-0 value.

These packets will define a subset of the available points. At the time of writing there are about 123 points defined. Combining the information in the 'point' packets will give the full set of available points.

The coordinates use a Z-Up, right-handed coordinate system.

2.7 Additional Information

These datagrams provide additional data, but do not by themselves define a full pose.



2.7.1 Joint Angles (type 20)

Information about each joint is sent as follows.

```
4 bytes point ID of parent segment connection. See 2.5.10  
4 bytes point ID of child segment connection. See 2.5.10  
4 bytes floating point rotation around segment x-axis  
4 bytes floating point rotation around segment y-axis  
4 bytes floating point rotation around segment z-axis
```

Total: 20 bytes per segment

The coordinates use a Z-Up, right-handed coordinate system.

2.7.2 Linear Segment Kinematics (type 21)

Information about each segment is sent as follows.

```
4 bytes segment ID See 2.5.9  
4 bytes x–coordinate of segment position  
4 bytes y–coordinate of segment position  
4 bytes z–coordinate of segment position  
4 bytes x component of segment global velocity  
4 bytes y component of segment global velocity  
4 bytes z component of segment global velocity  
4 bytes x component of segment global acceleration  
4 bytes y component of segment global acceleration  
4 bytes z component of segment global acceleration
```

Total: 40 bytes per segment

The coordinates use a Z-Up, right-handed coordinate system.

2.7.3 Angular Segment Kinematics (type 22)

Information about each segment is sent as follows.

```
4 bytes segment ID See 2.5.9  
4 bytes q1 rotation – segment rotation quaternion component 1 (re)  
4 bytes q2 rotation – segment rotation quaternion component 1 (i)  
4 bytes q3 rotation – segment rotation quaternion component 1 (j)  
4 bytes q4 rotation – segment rotation quaternion component 1 (k)  
4 bytes x component of segment global angular velocity  
4 bytes y component of segment global angular velocity  
4 bytes z component of segment global angular velocity  
4 bytes x component of segment global angular acceleration  
4 bytes y component of segment global angular acceleration  
4 bytes z component of segment global angular acceleration
```

Total: 44 bytes per segment

The coordinates use a Z-Up, right-handed coordinate system.



2.7.4 Motion Tracker Kinematics (type 23)

Information about each motion tracker is sent as follows.

```
4 bytes segment ID to which the tracker is attached See 2.5.9
4 bytes q_re tracker global orientation
4 bytes q_i tracker global orientation
4 bytes q_j tracker global orientation
4 bytes q_k tracker global orientation
4 bytes x-coordinate of tracker global free acceleration
4 bytes y-coordinate of tracker global free acceleration
4 bytes z-coordinate of tracker global free acceleration
4 bytes x component of segment local acceleration
4 bytes y component of segment local acceleration
4 bytes z component of segment local acceleration
4 bytes x component of segment local angular velocity
4 bytes y component of segment local angular velocity
4 bytes z component of segment local angular velocity
4 bytes x component of segment local magnetic field
4 bytes y component of segment local magnetic field
4 bytes z component of segment local magnetic field
```

Total: 68 bytes per segment.

Only data for segments with a tracker is sent. So it's important to check the segment ID for this datagram.

The coordinates use a Z-Up, right-handed coordinate system.

2.7.5 Center of Mass (type 24)

Information about the center of mass is sent as follows.

```
4 bytes x-coordinate of center of mass position
4 bytes y-coordinate of center of mass position
4 bytes z-coordinate of center of mass position
```

Total: 12 bytes

The coordinates use a Z-Up, right-handed coordinate system.

2.7.6 Time Code (type 25)

Information about time code is sent as follows.

```
12 byte string formatted as such HH:MM:SS.mmm
```

Total: 12 bytes



3 Data Types

3.1 Segment IDs

Table 1: Euler and Quaternion protocols

Segment Name	Segment Index	Segment ID
Pelvis	0	1
L5	1	2
L3	2	3
T12	3	4
T8	4	5
Neck	5	6
Head	6	7
Right Shoulder	7	8
Right Upper Arm	8	9
Right Forearm	9	10
Right Hand	10	11
Left Shoulder	11	12
Left Upper Arm	12	13
Left Forearm	13	14
Left Hand	14	15
Right Upper Leg	15	16
Right Lower Leg	16	17
Right Foot	17	18
Right Toe	18	19
Left Upper Leg	19	20
Left Lower Leg	20	21
Left Foot	21	22
Left Toe	22	23
Prop1	24	25
Prop2	25	26
Prop3	26	27
Prop4	27	28



Table 2: Unity3D protocol

Segment Name	Segment Index	Segment ID
Pelvis	0	1
Right Upper Leg	1	2
Right Lower Leg	2	3
Right Foot	3	4
Right Toe	4	5
Left Upper Leg	5	6
Left Lower Leg	6	7
Left Foot	7	8
Left Toe	8	9
L5	9	10
L3	10	11
T12	11	12
T8	12	13
Left Shoulder	13	14
Left Upper Arm	14	15
Left Forearm	15	16
Left Hand	16	17
Right Shoulder	17	18
Right Upper Arm	18	19
Right Forearm	19	20
Right Hand	20	21
Neck	21	22
Head	22	23

Client et serveur UDP pour linux

Déclaration du header pour le client et le serveur (commun au client et au serveur)

```
1 // UDP Client Server -- send/receive UDP packets
2 // Copyright (C) 2013 Made to Order Software Corp.
3 //
4 // This program is free software; you can redistribute it and/or modify
5 // it under the terms of the GNU General Public License as published by
6 // the Free Software Foundation; either version 2 of the License, or
7 // (at your option) any later version.
8 //
9 // This program is distributed in the hope that it will be useful,
10 // but WITHOUT ANY WARRANTY; without even the implied warranty of
11 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12 // GNU General Public License for more details.
13 //
14 // You should have received a copy of the GNU General Public License
15 // along with this program; if not, write to the Free Software
16 // Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
17 // 02110-1301 USA
18 #define SNAP_UDP_CLIENT_SERVER_H
19 #define SNAP_UDP_CLIENT_SERVER_H
20
21 #include <sys/types.h>
22 #include <sys/socket.h>
23 #include <netdb.h>
24 #include <stdexcept>
25
26 namespace udp_client_server
27 {
28     class udp_client_server_runtime_error : public std::runtime_error
29     {
30     public:
31         udp_client_server_runtime_error(const char *w) : std::runtime_error(w) {}
32     };
33
34
35     class udp_client
36     {
37     public:
38         udp_client(const std::string& addr, int port);
39         ~udp_client();
40
41         int get_socket() const;
42         int get_port() const;
43         std::string get_addr() const;
44
45         int send(const char *msg, size_t size);
46
47     private:
48         int f_socket;
49         int f_port;
50         std::string f_addr;
51         struct addrinfo * f_addrinfo;
52     };
53 }
```

```

53
54
55 class udp_server
56 {
57 public:
58     udp_server( const std::string& addr, int port);
59     ~udp_server();
60
61     int get_socket() const;
62     int get_port() const;
63     std::string get_addr() const;
64
65     int recv( char *msg, size_t max_size);
66     int timed_recv( char *msg, size_t max_size, int
67     max_wait_ms);
68
69 private:
70     int f_socket;
71     int f_port;
72     std::string f_addr;
73     struct addrinfo * f_addrinfo;
74 };

```

udp_client_server.h

Fichier c++ pour déclarer le nom d'espace udp_client_server (commun au client et au serveur) (le code est disponible ici : <https://drive.google.com/open?id=1kh37NmKNn9Afvk7SmhILJWCH-aqPy4-S>).

Code C++ pour le serveur sous linux

```

1 #include <string.h>
2 #include <unistd.h>
3 #include <errno.h>
4 #include <stdio.h>
5 #include "udp_client_server.h"
6 #include <iostream>
7
8
9 using namespace udp_client_server;
10 using namespace std;
11
12 int main( void )
13 {
14     char *msg;
15     udp_server* server = new udp_server("192.168.42.16", 9763);
16     server->recv( msg, 1000 );
17     cout << msg << endl;
18
19
20 return (0);
21 }

```

server.cpp

Code C++ du client sous linux

```
1 #include <string.h>
2 #include <unistd.h>
3 #include <errno.h>
4 #include <stdio.h>
5 #include "udp_client_server.h"
6 #include <iostream>
7
8 using namespace udp_client_server;
9 using namespace std;
10
11 int main( void )
12 {
13     char *msg;
14     int soc;
15     int iResult = 0;
16     udp_client* client = new udp_client("192.168.42.75", 9763);
17
18
19
20     soc = client->get_socket();
21     cout << soc << endl;
22
23
24     iResult = client->recv(msg, 1500);
25     if (iResult != -1)
26     {
27         cout << "Reception réussie du message ... \n";
28     }
29     else
30     {
31         cout << "Erreur n%d : Impossible de recevoir le message ... \n"
32             << errno;
33         return errno;
34     }
35
36     cout << "Le message est : %s \n" << msg << endl;
37
38 }
```

client.cpp

Pour compiler, il faut mettre trois fichiers dans le même dossier, à savoir udp_client_server.h, client_server.cpp et soit server.cpp, soit client.cpp. Il faut ensuite ouvrir un terminal, se placer dans le dossier puis exécuter g++ *.cpp (éventuellement avec des options d'erreurs). Il n'y a ensuite plus qu'à exécuter l'exécutable a.out produit.

Lecture de fichier .csv

```
1 #include<stdio.h>
2 #include<iostream>
3 #include <string>
4 #include <sstream>
5 #include <fstream>
6 #include<fstream>
7 #include <cstdlib>
8
9 using namespace std;
10
11 void parser (int lineNumberSought) {
12     ifstream file;
13     file.open("file.csv");
14     string value;
15     int c = 0;
16     string line, csvItem;
17     int lineNumber = 0;
18     float x;
19     while ( file.good() )
20     {
21         if (file.is_open()) {
22             while (getline(file,line)) {
23                 lineNumber++;
24                 if(lineNumber == lineNumberSought) {
25                     istringstream myline(line);
26                     while(getline(myline, csvItem, ',')) {
27                         switch (c){
28                             case 0:
29                                 cout << csvItem << endl;
30                                 c ++ ;
31                                 break;
32                             case 1 :
33                                 cout << csvItem << endl;
34                                 c ++ ;
35                                 break;
36                             case 2:
37                                 cout << csvItem << endl;
38                                 x=atof(csvItem.c_str())+0.5;
39                                 cout << x << endl;
40                                 c = 0 ;
41                                 break;
42                         }
43                     }
44                 }
45             }
46         }
47     }
48     file.close();
49 }
50
51 int main (void) {
52
53     int lineNumberSought = 3;
54     parser(lineNumberSought);
55     return 0;
56 }
57
```