Université de Bordeaux
Image & Son

Analyse & Visualisation de Trajectoires 3D

Projet de Fin d'Études

github

Auteurs :
Sofian Antri
Antonin Ayot
Adrien Célérier

Clients :
Arnaud Prouzeau
Vanessa P. Araya
Ambre Assor

Contents

1 Introduction

En informatique, la classification est un procédé utilisé pour regrouper des éléments et leur attribuer une classe ou une catégorie à l'aide d'un algorithme. Plus spécifiquement, lorsqu'on parle de classification non supervisée (ou *clustering*), on désigne une méthode regroupant ces éléments selon certaines caractéristiques prédéfinies. La classification non supervisée est la plupart du temps utilisée dans la recherche, mais peut aussi être utilisée dans d'autres domaines, par exemple en biologie ou en développement de logiciel et/ou d'application.

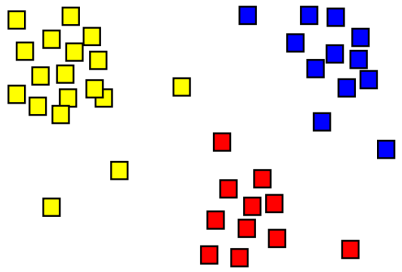


Figure 1: Exemple de *clusterisation* sur des carrés regroupés par position

Ici, l'objectif du projet consiste en l'analyse, la classification et la visualisation de trajectoires dans un espace 3D.
Pour ce faire, il s'agit dans un premier temps de collecter des données adéquates. En ce sens, les clients proposent une étude, réalisée par *Arnaud Prouzeau* dans le cadre de l'article Design and Evaluation of Interactive Small Multiples Data Visualisation in Immersive Spaces [1], visant à analyser les trajectoires empruntées par des sujets lors d'une expérience. Chaque sujet est équipé d'un casque de réalité virtuelle et doit prendre part à plusieurs tâches, chacune nécessitant des mouvements (et donc une trajectoire) au sein d'un environnement en 3D.
L'objectif est ensuite la classification des données collectées à l'aide d'un algorithme de *clustering*. K-Mean représente en ce sens le candidat idéal, étant un algorithme de classification facilement paramétrable par l'utilisateur et adaptable à des structures de données complexes telles des trajectoires.
Enfin, il s'agit de visualiser les résultats obtenus de manière logique, claire et précise. Dans le contexte de représentation visuelle de trajectoires 3D, certains moteurs 3D populaires satisfont amplement les besoins demandés. Ainsi, la bibliothèque IATK permet la création de représentations de données en haute qualité et de manière interactive via le moteur Unity 3D. Un tel outil sert donc de manière appropriée l'objectif du projet.

2 Contexte & analyse de l'existant

2.1 Algorithme de classification : K-mean

L'algorithme de classification implémenté dans le cadre du projet est l'algorithme K-Mean [2].
il s'agit d'un algorithme créé pour fonctionner avec un ensemble de points, sans nombre de dimensions fixe, ainsi qu'un entier k. L'algorithme va partitionner ces points selon k groupes, que l'on peut définir comme "clusters".
Par abus de langage, il se peut que les centres de cluster, ou représentation de cluster soient référé comme simplement "clusters".
L'algorithme se découpe en 2 parties : la phase d'initialisation et la phase d'analyse.

2.1.1 K-mean : Initialisation

Lors de la phase d'initialisation, on crée k points aléatoires dans l'espace correspondant à celui de l'ensemble de points, qui vont constituer le "centre" de chaque *cluster*, ou "point de moyenne".
Il existe une autre méthode, la méthode de Forgy, initialisant chaque centre de *cluster* sur un point aléatoire dans l'ensemble d'entrée.
Enfin, une dernière méthode possible est l'initialisation k-means++, qui essaye de répartir les centre de chaque *cluster* de la manière la plus optimale possible. On prend un point au hasard pour initialiser un centre de *cluster*, puis, pour le restes des centres, chacun est choisi parmi les points restants, avec une probabilité proportionnelle au carré de la distance entre le point et le *cluster* le plus proche.
La suite de l'algorithme dépend fortement de cette initialisation, et le fait qu'elle soit aléatoire peut rendre l'algorithme quelque peu inconstant.

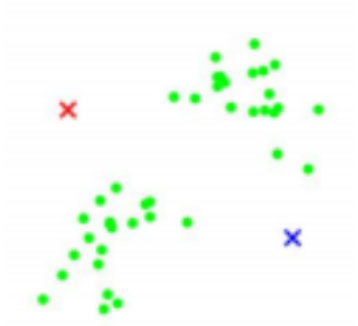


Figure 2: Initialisation de K-mean, suivant la méthode aléatoire, avec $k = 2$

2.1.2 K-mean : Analyse

La phase d'analyse comporte deux étapes, qu'elle répète en boucle jusqu'à ce qu'il y ait convergence, c'est-à-dire jusqu'à ce que ces étapes n'apportent plus de modifications. Ces deux étapes sont l'assignation (*assignment*) et la mise à jour (*update*).

L'étape d'assignation consiste à attribuer une responsabilité entre chaque *cluster* et chaque point. On remplit ainsi un tableau de responsabilités à double entrée, une pour les *clusters* et une pour les points. Ce tableau contiendra les valeurs 0 ou 1 : 1 si le *cluster* est celui le plus proche du point, 0 autrement.

$$\hat{k}^{(n)} = \underset{k}{argmin}\{d(m^{(k)}, x^{(n)})\}, \qquad r_k^{(n)} = \begin{cases} 1 & \text{if } \hat{k}^{(n)} = k \\ 0 & \text{if } \hat{k}^{(n)} \neq k \end{cases}$$

Ici r est le tableau de responsabilité, $\hat{k}^{(n)}$ est le cluster responsable du point à l'indice n, et d est la fonction d'heuristique utilisée pour calculer la distance entre le cluster et le point.

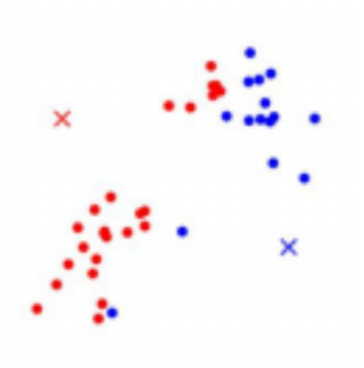


Figure 3: Étape d'assignation du K-mean

L'étape de mise à jour consiste à modifier les *clusters* afin qu'ils puissent mieux correspondre aux points dont ils sont responsables. Pour ce faire, la nouvelle représentation de chaque *cluster* deviendra la moyenne de tous les points dont il est responsable.

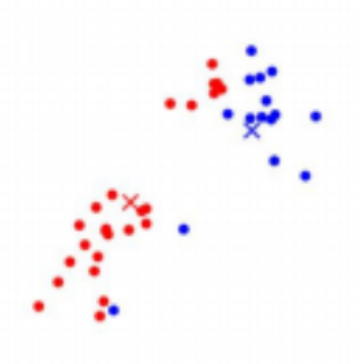


Figure 4: Étape de mise à jour du K-mean

Ces deux étapes tournent en boucle, jusqu'à convergence. La convergence est atteinte lorsque les représentations de *clusters* ne change pas lors de la mise à jour.

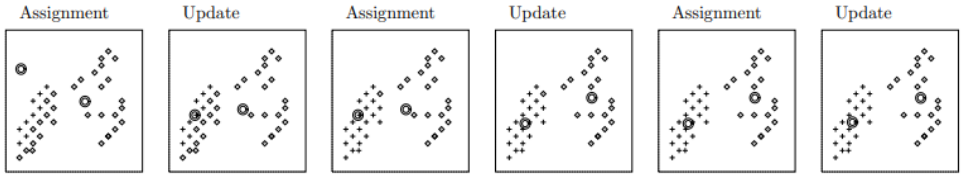


Figure 5: Exemple d'exécution de la phase d'analyse

2.1.3 Soft K-Mean

On dispose aussi d'une sorte de version alternative au K-mean, dite le "soft K-mean". L'algorithme est globalement le même, les modifications interviennent seulement dans la phase d'analyse. En remplissant le tableau de responsabilité, au lieu d'y insérer des 0 et des 1, on y inscrit des nombres entre 0 et 1, de cette façon:

$$r_k^{(n)} = \frac{\exp\left(-\beta\, d(\mathbf{m}^{(k)}, \mathbf{x}^{(n)})\right)}{\sum_{k'} \exp\left(-\beta\, d(\mathbf{m}^{(k')}, \mathbf{x}^{(n)})\right)}.$$

On remarque qu'un paramètre β a été ajouté, il correspond à la rigidité (*stiffness*) de l'algorithme. Ce paramètre varie de 1 à l'infini, et plus il est haut,

plus l'algorithme se rapprochera du K-mean normal.

2.1.4 Adaptation de l'algorithme

De base, K-mean est utilisé avec des points. Il va donc falloir adapter l'algorithme pour qu'il puisse *clusteriser* des trajectoires. Une approche possible pour cette transition est d'analyser les trajectoires point par point.

2.2 Base de données

Il est nécessaire de se baser sur des données solides afin de garder une ligne de conduite constante tout au long du projet. Les clients ont donc proposé une étude réalisée par *Arnaud Prouzeau* dans le cadre de la rédaction d'un article [1] sur les comportements vis-à-vis de représentations de données dans un espace immersif.
Dans cette étude, des participants s'attelaient à plusieurs reprises à différentes tâches à l'aide d'un casque de réalité virtuelle. Leurs mouvements effectués lors de ces expériences peuvent être retranscrits en tant que trajectoires.

Les résultats de cette étude sont composés de tableurs CSV, notamment un spécifique représentant la position dans l'espace au cours du temps du casque VR pour chaque participant.
Ces CSV seront très utiles puisque composés de dizaines de milliers de points 3D, regroupés selon différent types de tâches effectuées, et donc une base idéale de *clusters*.

Plus précisément, les participants devaient évoluer au sein de trois espaces virtuels, mettant en scène des représentations de données en 3D différentes.
La représentation **Flat** consistait en l'alignement des données à visualiser sous la forme d'un "mur" droit.
La représentation **Half-Circle** consistait quant à elle à disposer les données en arc de cercle autour du participant.
Enfin, la représentation **Full Circle** était, comme son nom l'indique, un cercle de données réparti autour du participant.

On peut attendre de ces différentes tâches des types de trajectoires produites par les participants différents, ce qui fait donc bien de cette étude une base de données idéale pour de la *clusterisation*.

2.3 Visualisation de trajectoires 3D

2.3.1 Unity 3D

Unity est une application mettant en avant un moteur 3D. Il est surtout réputé pour être utilisé dans le domaine du jeu-vidéo, mais s'est retrouvé à être utilisé dans la recherche pour sa facilité d'utilisation et sa compatibilité avec des logiciels externes. C'est sur cette application que l'affichage des *clusters* s'effectuera.

2.3.2 IATK

IATK (*Immersive Analytics Toolkit*) [3] est un module de Unity 3D développé par *Maxime Cordeil*, *Andrew Cunningham* et *Benjamin Lee* qui permet de représenter des données complexes (nuages de points, courbes, matrices... etc.) de manière interactive au sein d'une scène. Pour ce faire, il s'agit de créer un objet **Visualisation** qui, assorti d'un script, propose de nombreuses variables à paramétrer (couleur, géométrie... etc.) vis-à-vis d'une base de donnée quelconque. Cette base de données doit être renseignée depuis un second objet, **Data Source**, qui permet notamment de charger des fichiers CSV.
Ces deux objets représentent donc des outils très utiles pour la partie visualisation du projet.

2.3.3 Autre

On connait également l'existence d'un module similaire qui a été créé du nom de FiberClay, qui permet de représenter des données sous formes de nuages de points, de courbes, ou sous forme de traînée de points. L'outil est interactif et l'utilisateur peut sélectionner les données qu'il veut mettre en valeur afin de les afficher en surbrillance.

3 Besoins

3.1 Besoins fonctionnels

Le premier besoin est un algorithme qui permet de "*clusteriser*" des trajectoires, c'est-à-dire regrouper selon des ressemblances. La manière de regrouper ces données doit également être discutée.

Il faut également un outil qui va non seulement permettre de visualiser les trajectoires en 3 dimensions, mais également nous permettre de les visualiser selon le *cluster* auquel elles appartiennent, de manière assez claire et compréhensible.

Idéalement, le produit délivré doit pouvoir être paramétré selon les désirs de l'utilisateur. Étant donné que les données utilisées sont assez irrégulières, représentant des trajectoires qui ont un nombre de points indéfini, et souvent assez grand, il est nécessaire de réduire la quantité de données, et l'utilisateur doit pouvoir choisir à quel point il veut la réduire. De même, l'algorithme K-Mean doit aussi pouvoir être paramétré, afin de pouvoir regrouper les trajectoires selon la méthode choisie par l'utilisateur.

3.2 Besoins non-fonctionnels

Le produit se doit d'avoir une vitesse adaptable. L'algorithme K-mean n'est pas long à exécuter, vu qu'il a tendance à converger assez vite.
Cela dit, sa rapidité est dépendante du nombre de points. C'est pour cela que la minimisation des données doit être implémentée comme paramétrable. On peut ainsi avoir une quantité importante de données si l'on accepte de passer du temps à les traiter, ou à l'inverse, avoir une quantité de données très limitée si l'on veut un calcul rapide.
Ce qui, en revanche, peut prendre du temps, c'est la minimisation des données. En effet, en tant que paramètres de K-Mean, les trajectoires doivent toutes avoir la même longueur, ce qui nécessite donc un ajustement supplémentaire.

Concernant la rapidité d'affichage, la base est dépendante d'Unity et du module IATK. En effet, lorsque la quantité de données données est trop importante, le temps de chargement peut s'avérer particulièrement long. Il s'agit donc de retirer des données qui ne sont pas utilisées lors du chargement de la base de données.

Pour ce qui est de la maintenabilité du produit, l'architecture créée doit permettre à d'autres développeurs de pouvoir apporter des changements ou corrections de manière assez aisée.

4 Discussion

Il s'agit maintenant de discuter des différentes méthodes et approches qui ont été pensées, et implémentées pour la plupart, pour le coté algorithmique de ce projet.

4.1 Minimisation

La première partie à aborder est la minimisation, ce qui permet de réduire le nombre de points des trajectoires, afin de rendre les calculs plus rapides. Trois approches ont été implémentées pour cette méthode.

Pour commencer, une méthode très simple et très basique a été testée en début de projet, qui consistait à enlever un point sur 3, 4, ou une quelconque autre valeur. Cette méthode a été faite manuellement sur un tableur et n'a pas été implémentée, étant donné qu'elle était très basique et pouvait facilement entraîner de la perte d'information.

La deuxième idée consiste en la seule conservation des points de la trajectoire marquant un changement de direction assez prononcé. La méthode est la suivante : prenons un exemple avec trois points A, B et C dans une trajectoire. Si la direction du point A au point B est positive sur les axes x, y et z, et qu'il en est de même pour la direction de B à C, alors B peut être retiré. À l'opposé, si les signes ne sont pas identiques pour la direction AB et la direction BC, alors il faut garder B car il marque un changement de direction. Cette méthode a néanmoins comme points faibles d'ignorer certains angles importants (dans le pire des cas, certains peuvent se rapprocher de 90° sans changer de signe sur un quelconque axe) et, à l'inverse, de prendre en compte des angles très faibles (proches de la ligne droite, mais pouvant quand même provoquer un changement de signe).

Enfin, la méthode optimale choisie pour cette problématique adopte une stratégie permettant de retirer les points ne créant pas de différence importante le long de la trajectoire. Cette méthode nécessite un paramètre, qui est le ratio (également une limite, que l'on verra à la section suivante). Ce ratio servira à déterminer si la longueur a été suffisamment peu changée pour que le point soit retiré. Encore une fois, prenons l'exemple avec les points A, B et C, et un ratio de 0,98. Si la distance AC est supérieure à $0,98 * (AB + BC)$, alors le point peut être retiré, car la modification de distance est suffisamment moindre pour que l'on puisse se permettre de modifier cette information. Cette méthode parcourt la trajectoire point par point, et peut enlever au maximum la moitié des points sur une itération.

4.2 Harmonisation

Après la minimisation vient l'harmonisation, qui permet d'avoir des trajectoires avec le même nombre de points. Cela est notamment nécessaire dans le cas de l'utilisation de l'algorithme K-Mean.

La fonction d'harmonisation utilisée est faite pour fonctionner de paire avec la dernière méthode de minimisation décrite, c'est-à-dire une fonction qui peut doit être appelé sur plusieurs itérations.

Cette fonction prend un ratio (vu précédemment) et une limite en paramètres. Cette limite sert à déterminer le nombre maximum de point que l'on veut dans nos trajectoires (on peut rentrer 0 pour minimiser les trajectoires le plus possible).

Grossièrement, cette fonction contient une boucle, et applique une minimisation sur la trajectoire la plus longue à chaque tour de boucle, et s'arrête lorsque toutes les trajectoires ont atteint la limite.

Si la limite n'est pas encore atteinte lorsque l'on ne peut plus retirer de point sur une trajectoire, la limite est rehaussée pour correspondre à la taille de cette dernière.

Le seul problème de cette méthode est qu'elle prend du temps, car, si jamais trop de points sont retirés d'une trajectoire et que la limite fixée est dépassée dans le processus, la trajectoire doit être réinitialisée, car il n'est pas possible de lui rajouter des points de manière quelconque, et garder tous les états de chaque trajectoire ne serait pas efficace.

4.3 Heuristique et représentation interne des trajectoires

Une des problématiques importantes était la représentation interne des trajectoires avant analyse et, étant donné que le K-Mean calcule les distances entre les éléments pour les regrouper, comment comparer ces trajectoires, et leur attribuer un score selon leur distance.

Pour ce faire, une méthode simple a été tout d'abord implémentée : la distance euclidienne point par point. Cette méthode, bien que rudimentaire, est efficace pour regrouper les trajectoires qui sont proches les unes des autres en terme de distance, mais la forme de la trajectoire n'est pas réellement prise en compte, il a donc du fallu implémenter quelque chose de plus approprié.

Il a ensuite s'agit d'implémenter une variante de la méthode précédente, qui se base toujours sur la distance euclidienne point par point, mais qui, avant ce calcul, ramène toutes les trajectoires à une origine commune. cela permet de mettre de côté la distance au sein du repère, et de s'intéresser davantage à la direction globale des trajectoires.

La dernière méthode implémentée est une méthode de comparaison vecteur par vecteur au lieu de point par point. Cette méthode s'avère efficace, mais elle

partage les défauts des autres méthodes évoques ci-après.

Un des problèmes majeurs de ces méthodes est le traitement des données représentées par la trajectoire. Une trajectoire est un objet complexe, la séparer point par point (ou vecteur par vecteur) rend l'information plus simple à traiter, mais aussi très approximative.

Un autre défaut inhérent aux méthodes point par point est celui amené par la répartition des points au sein des trajectoires. Il est par exemple possible d'avoir deux trajectoires visuellement très similaires en terme de forme, mais avec des répartitions de points très différentes, ce qui impliquerait que, avec les méthodes de calcul de distance utilisées, elles obtiendraient un score assez bas.

De plus, le calcul point par point ne marche pas forcément avec la minimisation/harmonisation décrite précédemment selon l'interprétation que l'on souhaite attribuer aux données. En l'occurrence, le facteur de temps entre chaque point a été ignoré pour rendre la comparaison moins complexe.
Dans le cadre de l'étude de *Arnaud Prouzeau*, il aurait été possible de comparer les trajectoires en utilisant les positions de chaque participant à plusieurs instants donnés, mais cela soulèverait beaucoup de problèmes d'interprétations. Le problème majeur étant la comparaison entre les trajectoires d'utilisateurs terminant à des temps différents, problème qui s'épaissit encore plus si leurs trajectoires sont similaires. Beaucoup de solutions arbitraires sont envisageables et applicables au cas-par-cas, ce qui explique la décision d'ignorer l'information du temps.

La comparaison des trajectoires est la pierre angulaire de l'algorithme K-Mean, il faudrait donc un moyen optimal de comparer les trajectoires dans leur globalité pour obtenir une *clusterisation* la plus efficace possible.

4.4 Algorithme K-mean

Pour rappel, l'algorithme se découpe en trois parties : initialisation, assignation et mise à jour.

Pour ce qui est de l'initialisation, plusieurs méthodes ont été implémentées, avec comme point commun l'utilisation des trajectoires de la base de données:

- la première est simplement de prendre le nombre souhaité de trajectoires au hasard dans la base de données,
- la deuxième est de prendre les premières trajectoires représentant chaque type de layout, à savoir Flat, Half Circle et Full Circle dans le cadre de l'étude proposée par les clients,
- la troisième prend le nombre souhaité dans chaque layout comme la précédente, mais aléatoirement.

Une amélioration a été apportée à cette initialisation, en décalant légèrement les trajectoires représentant les clusters de manière aléatoire pour chacun de leurs points.
Prendre des trajectoires aléatoires en dehors de la base de données aurait été plus compliqué, mais aussi possiblement moins efficace. Il faudrait pouvoir générer aléatoirement des trajectoires cohérentes, et ensuite il faudrait qu'elles se rapprochent de trajectoires déjà existantes, cela paraît donc être laborieux et peu efficace.

Pour ce qui est de l'assignation et de la mise à jour, aucune modification n'a été nécessaire, l'algorithme est le même qu'en théorie, de même que pour le fonctionnement global de l'algorithme.
Il peut cependant être intéressant de noter qu'afin d'éviter d'effectuer trop de manipulations à chaque comparaison de trajectoires, ces dernières sont modifiées dès le début (ramenées à l'origine, ou transformée en liste de vecteurs).

5 Architecture du projet

5.1 Traitement des trajectoires

Les trajectoires seront introduites dans une classe spécifique pour le traitement de celles-ci. Cette classe comportera une liste contenant les trajectoires, une liste contenant le type de layout propre à chaque trajectoire contenue dans la classe (Flat, Half-Circle ou Full-Circle dans le cadre de l'étude propsoée par les clients) ainsi qu'une autre liste permettant de savoir quel utilisateur a effectué telle trajectoire.

La classe comporte des méthodes permettant :

- d'ajouter une trajectoire manuellement, de manière aléatoire ou à partir d'un fichier CSV,
- d'effectuer une translation sur les trajectoires pour qu'elles aient la même origine,
- de passer les trajectoires en représentation vectorielle et inversement,
- d'écrire les trajectoires dans un fichier CSV,
- d'harmoniser et de minimiser les trajectoires,
- d'afficher les trajectoires (pour le debug).

Ces fonctions seront à exploiter pour effectuer la classification des trajectoires.

5.2 Classification des trajectoires

Cette classification se fera dans un fichier autre que celui des traitements des trajectoires. C'est ce fichier qui effectuera l'algorithme K-Mean selon le document présenté par les clients [2].

Cet autre script retournera, selon les trajectoires données et les paramètres relatifs à l'algorithme, des tableaux qui représenteront le *cluster* d'appartenance pour chaque trajectoire, ainsi que les noms des fichiers créés pour les trajectoires minimisées.

5.3 Communication des données

Il s'agit de la partie concernant le transfert de données entre Python et Unity. Plus précisément, de Python à Unity pour les résultats fournis par l'algorithme de K-Mean, et d'Unity à Python pour l'interaction avec l'utilisateur. Afin de faciliter la communication entre l'application Unity et les scripts Python,

l'utilisation de sockets est idéale, ceux-ci utilisant le protocole TCP/IP qui est compris quelque soit le langage.

Dans un premier temps, l'utilisateur entrera les paramètres qu'il souhaite sur Unity (fichiers à charger, nombre de *clusters*... etc.). Ensuite, lorsque l'application se lance et que l'utilisateur lance le transfert, Unity exécutera le script Python automatiquement et enverra les paramètres sockets. Python de son côté effectuera tout le processus de minimisation des trajectoires et du K-Mean. Une fois celui-ci terminé, le script envoie à Unity les résultats du K-Mean et les trajectoires minimisées.

Cette méthode permet de gérer une grande partie de calcul et du parsing de grands fichiers CSV sans utiliser Unity, qui est moins efficace que certains langages et bibliothèques (par exemple Numpy dans le cas de Python).

5.4 Visualisation sur Unity 3D

La bibliothèque IATK propose déjà la visualisations de trajectoires indépendantes. Il s'agit donc de créer un module permettant de regrouper visuellement ces trajectoires par *cluster* d'appartenance.
En ce sens, un objet (prefab) sera créé, permettant la saisie des paramètres par l'utilisateur et l'exécution du script de communication des données. Ce script permettra également de parser les données récupérées et de les afficher de manière claire dans une scène.

6 Implémentation

6.1 Python : Minimisation et clusterisation

Python a été le choix idéal car ce langage interprété permet d'avoir un résultat efficace et d'opérer très rapidement sur une quantité de données importante (les fichiers CSV contenant les trajectoires par exemple) via des bibliothèques comme Numpy.

6.1.1 Gestion des trajectoires

Le premier script, **trajectory_clustering.py**, est celui de la gestion des trajectoires. Il implémente une classe *Trajectories* qui stocke toutes les trajectoires dont l'utilisateur a besoin pour effectuer la classification. La représentation peut se faire soit avec des points, soit avec des vecteurs.

Les trajectoires sont stockées dans un tableau d'entiers à 3 dimensions. La première dimension liste chacune des trajectoires, tandis que la deuxième dimension est consacrée aux points et la seconde aux coordonnées de chaque point. Le type de *Layout* est une chaîne de caractères représentant le type de la trajectoire. Et enfin la liste des utilisateurs qui est une liste d'entiers déterminant quel utilisateur a effectué telle trajectoire.

L'affichage utilisé pour le debug est produit via la bibliothèque *Matplotlib* qui permet facilement et efficacement de représenter des données sous forme graphique.

6.1.2 Application de l'algorithme K-Mean

Le script implémentant l'algorithme du K-Mean est **k_mean.py**. Comme décrit en plus haut, il contient en majorité les fonctions représentant chaque partie de l'algorithme. L'initialisation (fonction *initializationFromTrajectories()*) a pour but de créer un objet *Trajectories* contenant les trajectoires faisant office de *clusters*. L'utilisateur peut décider grâce aux différents paramètres si il veut un cluster par Layout ou un cluster aléatoire.
L'assignation est divisé en différentes fonctions :

- *assignment()* qui est la fonction principale de cette partie. C'est celle-ci qui va calculer les prédictions de l'algorithme K-Mean,
- *softAssignment()* qui est la version de la fonction *assignment()* pour le Soft K-Mean.

Et enfin une autre fonction *update()* qui va mettre à jour les clusters en fonction des trajectoires leur appartenant. De plus, ce script contient le K-Mean basique ainsi que sa version Soft K-Mean (que l'utilisateur peut choisir depuis Unity)

et la version crée par OpenCV. Cette dernière a eu pour but de comparer les résultats de l'algorithme K-Mean implémenté par le script avec celui proposé par OpenCV.

6.1.3 Réception et envoi des données

Ce script utilise des sockets afin de pouvoir se connecter à l'application. Il est implémenté de manière à recevoir d'abord les informations depuis Unity, puis d'effectuer le calcul du K-Mean avec les fichiers rentrés en paramètres et enfin de renvoyer les résultats à Unity.

Cette tâche est effectué par le script **socket_sender.py**. Deux fonctions ont été implémentées pour faciliter la compréhension du code :

- *getMessageFromUnity()*, qui récupère une information depuis Unity,
- *sendMessageToUnity()*, qui à l'inverse envoie une information à Unity.

Une fois le script lancé, celui-ci se connecte avec les deux arguments donnés, respectivement l'adresse IP et le port. Il récupère tous les paramètres rentrés par l'utilisateur sur Unity puis commence à charger tous les fichiers reçus. Ces fichiers (qui contiennent les trajectoires) sont ensuite donnés à une instance de la classe *Trajectories* afin de pouvoir charger les trajectoires dans celle-ci. Le script minimise et harmonise les trajectoires avant d'appeler la fonction de K-Mean. Il appelle ensuite une fonction de la classe *Trajectories* permettant d'enregistrer les trajectoires minimisées au format CSV, pour le chargement de ceux-ci sous Unity. Cela permettra par la suite de charger les trajectoires en peu de temps pour la visualisation.

Le script termine par envoyer le tableau d'assignation des *clusters* ainsi que les noms des fichiers des trajectoires minimisées à Unity pour la partie visualisation.

6.2 Unity : Interaction et affichage

Côté Unity, nous utilisons la bibliothèque IATK pour avoir un affichage 3D des trajectoires. Il faut de préférence d'abord minimiser les trajectoires sous peine d'avoir un chargement trop long sur Unity. Notre solution serait donc de créer un script C# qui serait capable d'envoyer les paramètres saisis, au préalable par l'utilisateur, au script Python. De plus, afin de faciliter l'interaction avec l'utilisateur, le script C# doit lancer le script python automatiquement.

6.2.1 Connexion avec Python

Tout ceci est le rôle du script **KMeanScript.cs**, qui permet à l'utilisateur d'entrer les différents paramètres essentiels à l'algorithme K-Mean (Fichiers à

charger, nombre de *clusters*... etc.) ainsi que des paramètres permettant le bon fonctionnement du script (chemin vers l'application Python ou port du socket). Il est important de noter que l'utilisateur doit impérativement avoir un environnement Python ainsi que les bibliothèques propres au bon fonctionnement du script.

Au sein de la classe du script, plusieurs variables sont mises à disposition en *public* afin que l'utilisateur puisse y interagir directement via l'inspecteur d'Unity. C'est avec ces attributs que l'utilisateur saisira les paramètres pour le K-Mean.

Une fois l'application Unity lancée, l'utilisateur devra appuyer sur la touche Espace pour lancer l'acquisition des données avec les paramètres fournis. Il est bon de noter qu'une telle implémentation devrait permettre à l'utilisateur de relancer une *clusterisation* avec de nouveaux paramètres sans avoir à relancer la scène. Lorsque la touche est appuyée, le script Unity lance automatiquement le script Python et par la même la connexion avec le socket côté Unity. Le script Unity va donc se charger d'envoyer les paramètres de l'utilisateur à Python et se mettre en attente des résultats que Python va lui fournir.

Une fois l'acquisition terminée, Unity ferme le socket et se met en attente d'une autre acquisition.
Le script possède désormais les paramètres nécessaire à la visualisation.

6.2.2 Visualisation

Une fois les résultats du script Python réceptionnés, il s'agit de les parser. Le tableau d'assignation des *clusters* est parsés sous la forme d'une liste de liste d'entiers, chaque sous-liste représentant les affectations à des *clusters* pour chaque trajectoire d'un fichier. Dans le cas classique d'une trajectoire par fichier, la liste sera composée de sous-listes de taille 1.
Le tableau contenant les noms des fichiers est parsé sous la forme d'une liste de chaînes de caractères.

Suivant ces listes, le prefab K-Mean sera rempli d'autant de prefabs Trajectory (composé d'une Data Source IATK et d'une Visualisation IATK) qu'il y a d'entiers dans le tableau d'assignations.
Chacune de ses trajectoires se voit affecter une couleur en fonction du *cluster* auquelle elle appartient. Chaque nom de fichier est chargé en tant que CSV dans le Data Source respectif. Les paramètres de la visualisation IATK comme les coordonnées (axes X, Y & Z) ou la géométrie (lignes reliant les points) et sont ensuite mis à jour afin d'afficher proprement chaque trajectoire.

Concernant l'organisation dans l'espace 3D des repères de trajectoires, les membres d'un même *cluster* sont alignés horizontalement depuis un point d'origine qui incrémente verticalement pour chaque nouveau *cluster*.

7 Tests

7.1 Tests de l'algorithme

Cette implémentation du K-Mean a dû être testé sur la performance et sur la fiabilité des résultats. Pour le 1er cas, certains paramètres existent pour en tester les limites, comme le nombre d'itérations. Le K-Mean s'arrête automatiquement dès que le tableau d'assignation ne change plus. En plus de l'algorithme K-Mean, il faut prendre en compte la minimisation des trajectoires avant le lancement de l'algorithme, qui s'effectue en temps exponentiel selon le nombre de points. Plus les trajectoires seront denses et plus la minimisation prendra du temps. Hors minimisation (qui est non-comprise dans le K-Mean) l'algorithme s'effectue en temps raisonnable.

Concernant la fiabilité des résultats, une visualisation hors Unity était nécessaire pour détecter quelle trajectoire appartient à quel *cluster*. Un affichage via la bibliothèque MatPlotLib a donc donc été rajouté pour en tester les résultats. Ce que l'algorithme donne est cohérent avec les trajectoires en entrée, malgré l'approximation lors de l'étape d'assignation. L'optimisation Soft K-Mean permet d'obtenir de meilleurs résultats.

7.2 Comparaison avec d'autres implémentations

Le K-Mean étant implémenté sans inspiration externe (hormis [2]) dans les scripts python, une autre implémentation a été importée pour y tester sa fiabilité. Cette autre implémentation est celle d'OpenCV. Les paramètres à rentrer son relativement similaires à la version implémentée dans les scripts, à savoir les trajectoires, le nombre de K-Mean et le nombre d'itérations. Le reste est composé de paramètres de critère qui sont propre à la bibliothèque OpenCV. Les résultats obtenus sont similaire à l'implémentation fournie par les scripts Python.

7.3 Étude des résultats

Les résultats affichés avec matplotlib sont disponible en annexes. On peut constater que les trajectoires dans le layout Full Circle sont le plus souvent regroupées ensemble, et de manière un peu moins efficace, le phénomène se produit aussi pour le layout Flat. En revanche on peut observer que pour le layout Half-Circle, on obtient souvent un mélange un peu chaotique des deux autres en terme d'appartenance aux clusters. Ce phémonème semble explicable de par le fait que ce layout spécifique est en quelque sorte un mélange hybride des deux autres, entre le "mur" plat et le cercle complet. Il est également intéressant de remarquer que l'implémentation du soft K-Mean a servi à améliorer les résultats de l'algorithme.

La représentation de l'implémentation OpenCV présente des résultats similaires, ce qui permet de déduire que les résultats de l'algorithme implémenté pour ce projet sont satisfaisants.

Enfin, les représentations dans une scène en 3D obtenues sur Unity permettent de naviguer directement au sein des trajectoires et de les observer sous différents angles. À tel point que de simple captures d'écran ne retranscrivent pas idéalement cette méthode de visualisation, bien plus efficace en lançant réellement la scène.
Toutefois, on peut observer que les trajectoires peuvent bien être *clusterisées* avec des paramètres de K-Mean différents, ici avec 5 *clusters* initialisés aléatoirement, ou encore 1 *cluster* par layout.

8 Tenue du projet

Pendant toute la durée du projet, nous avons tenté de nous atteler au diagramme de Gantt prévisionnel ci-dessous :

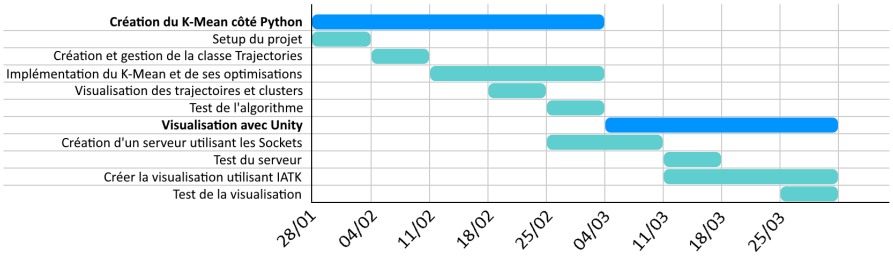


Figure 6: Gantt Prévisionnel

Certaines tâches s'avéraient être plus longues que prévu. La création et la gestion de la classe Trajectories a dû être mis à jour au fur et à mesure de l'avancement de la tâche d'implémentation du K-Mean. L'une des causes principales étant l'heuristique utilisée : nous avons dû en implémenter plusieurs pour rendre notre K-Mean efficace et l'adapter au calcul pour les trajectoires (cet algorithme étant prévu à la base pour des nuages de points). La tâche de visualisation des trajectoires étant plus ou moins liée à celle de la gestion de la classe, son temps a été rallongé en conséquence.

Concernant l'implémentation du K-Mean, celui-ci ainsi que ses optimisations ont été plus long à ajouter que prévu. De même que pour la tâche de la classe Trajectories, différentes implémentation ont dû être testé pour garder la plus efficace.

Ces rallongements nous ont conduit à raccourcir les tâches visant la partie visualisation sur Unity, ce qui nous mène au diagramme de Gantt Effectif suivant:

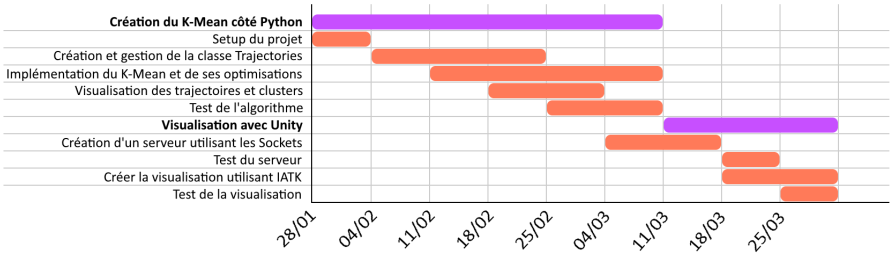


Figure 7: Gantt Effectif

Ces retards dans l'organisation nous ont empêché d'effectuer plusieurs autres tâches, notamment en ce qui concerne la représentation des trajectoires, la flexibilité de notre algorithme ou même la comparaison du K-Mean avec d'autres algorithmes de clustering.

9 Améliorations & Corrections possible

Le produit rendu n'est pas complètement fini, et des améliorations sont à prévoir pour obtenir un produit se rapprochant encore davantage des attentes des clients.

9.1 Flexibilité des bases de données

Le script Python implémenté est très dépendant de la base de données obtenue par l'étude découlant de l'article Design and Evaluation of Interactive Small Multiples Data Visualisation in Immersive Spaces [1]. Les fichiers CSV pris en entrée doivent contenir les mêmes colonnes que ceux de cette étude, autrement le bon déroulement de l'exécution peut s'en retrouver compromis.

9.2 Représentation et interprétation des trajectoires

Comme discuté plus haut, l'approche pour l'interprétation des données reste imparfaite et pourrait être améliorée. Le calcul des distances est un pré-requis majeur pour l'algorithme k-mean, il faudrait donc pouvoir analyser les trajectoires dans leur globalité plutôt que de les analyser point par point.

9.3 Nombre de trajectoires par fichier en sortie du script Python

Actuellement, le script Python crée un nombre de fichiers égal à celui de trajectoires, c'est le cas classique on l'on veut que chaque graphe affiché sous Unity contienne une seule trajectoire. Il faudrait ajouter une fonctionnalité pour pouvoir regrouper les trajectoires dans les fichiers selon des critères particuliers comme le *cluster* d'appartenance, le participant ou la tâche effectuée.

9.4 Comparaison avec d'autres algorithmes de classification

Afin de tester les performances de l'algorithme du K-Mean, comparer celui-ci avec d'autres algorithmes de clusterisation aurait été pertinent pour la discussion de la méthode choisie sur les performances et les résultats. L'utilisation

d'un autre algorithme que celui du K-Mean aurait probablement permis d'avoir de meilleurs résultats ou même de meilleures performances.

9.5 Améliorations visuelles

L'affichage sur Unity n'est pas aussi clair que désiré. Les courbes sont affichées selon leur *cluster* de référence, mais rien ne permet textuellement d'identifier ces données et de savoir à quoi elles correspondent. Il faudrait ajouter les informations du participant, de la tâche et de l'épreuve en cours, ainsi que le layout utilisé. Il pourrait également être envisagé d'écrire les centres de cluster dans un/des fichiers à part pour pouvoir les charger et les visualiser sous Unity.

9.6 Chargement dynamique de nouveaux fichiers

Actuellement, lors d'une seule exécution du script Unity, l'utilisateur ne peut pas charger plusieurs fois des fichiers en appuyant sur la barre Espace. Cela est dû au fait que la fonctionnalité n'a pas été envisagée avant la fin du projet. cela constituerait davantage un ajout envisageable qu'une correction.

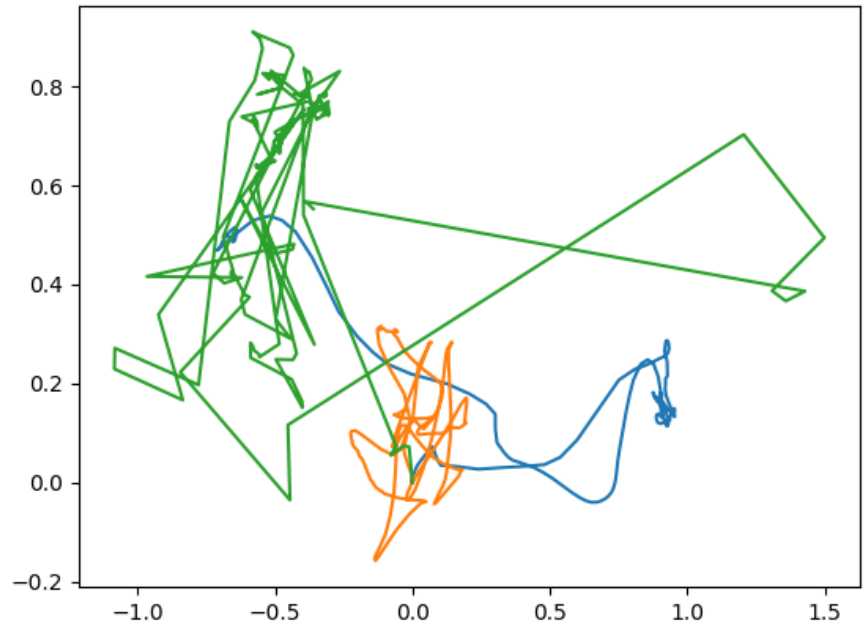
10 Conclusion

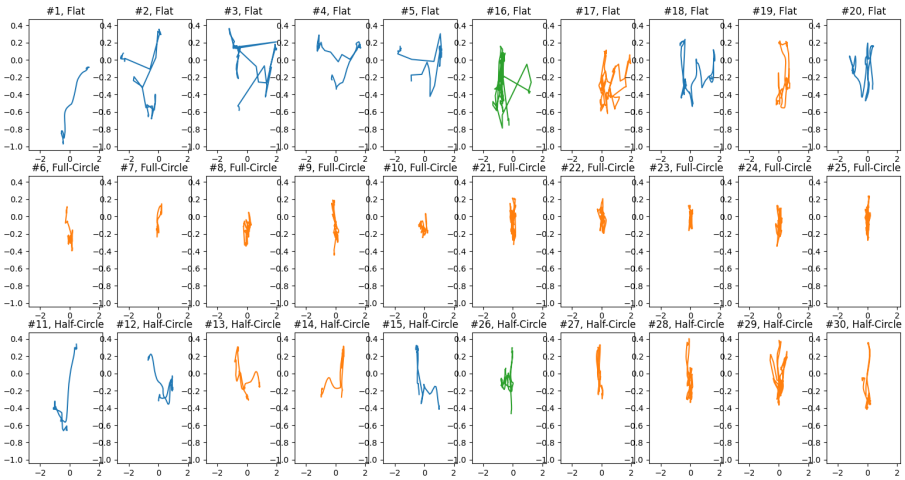
Par l'intermédiaire de ce projet, il a été possible de mettre en place une classification de trajectoires et leur visualisation dans un repère en 3 dimensions via différents outils comme MatPlotLib ou Unity 3D. L'utilisateur lamba pourra ainsi analyser ses propres bases de données via les outils proposés et les analyser comme bon lui semble.

Néanmoins, comme discuté en profondeur dans la section précédente, plusieurs pistes d'améliorations sont envisageables.
À cela pourraient être envisagées de nouvelles méthodes, comme par exemple l'utilisation du Deep Learning pour la classification. Cela serait pertinent dans le sens où un tel outil peut s'avérer très performant pour classifier des données complexes avec lesquelles les méthodes de comparaison restent incertaines.
Il peut aussi être fait mention de l'étendue de la visualisation à la réalité virtuelle, en utilisant d'autres modules Unity comme VRTK, permettant à un utilisateur d'explorer la scène 3D à l'aide d'un casque de réalité virtuelle.

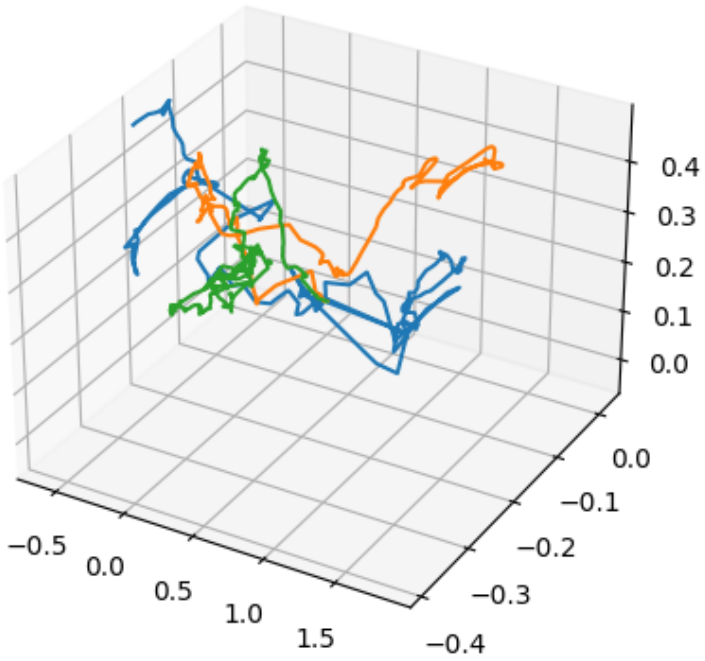
11 Annexes

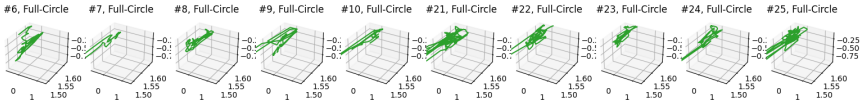
ANNEXE A : K-Mean sur des trajectoires en 2D

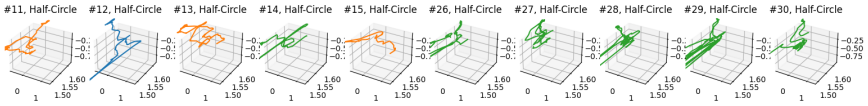


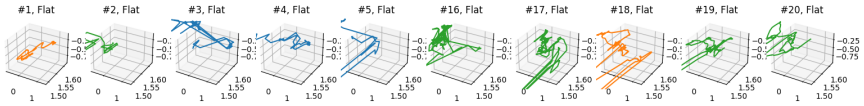


ANNEXE B : K-Mean avec 3 clusters aléatoires

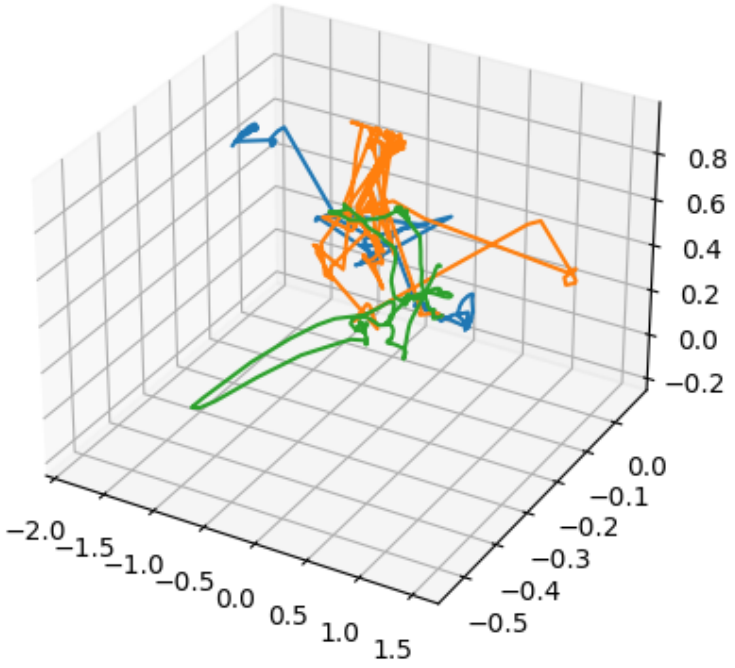


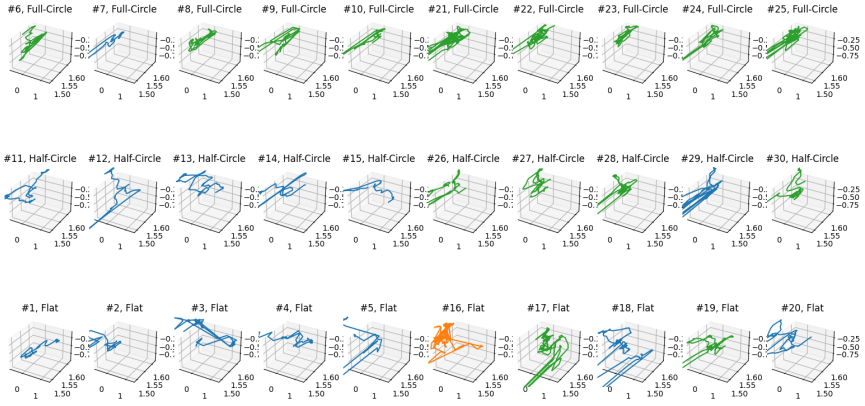




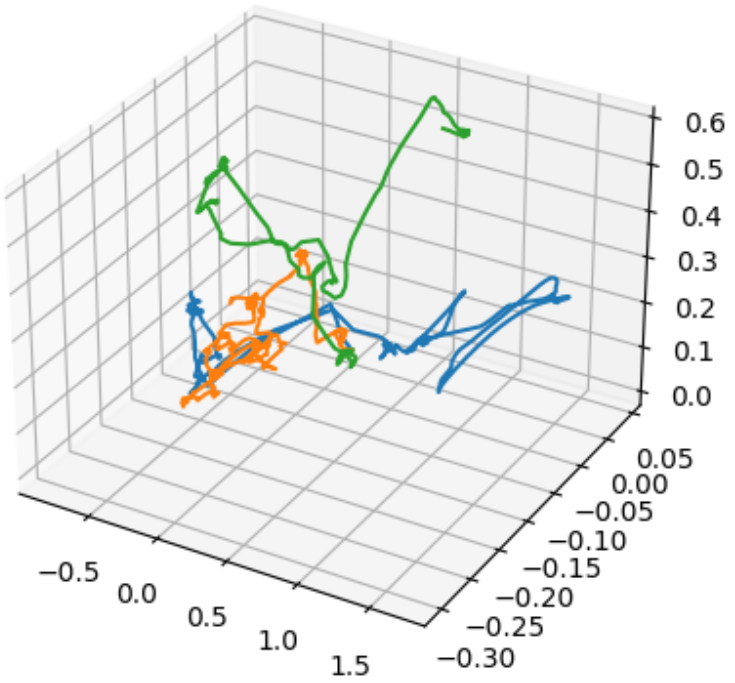


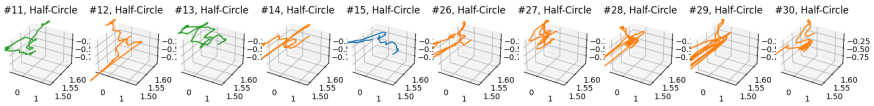
ANNEXE C : Soft K-Mean avec 3 clusters aléatoires

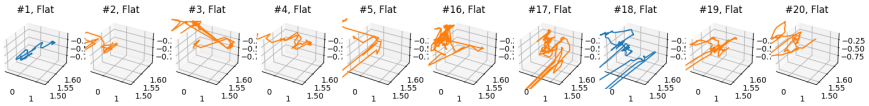


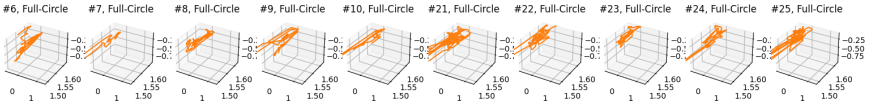


ANNEXE D : K-Mean avec 1 cluster par layout

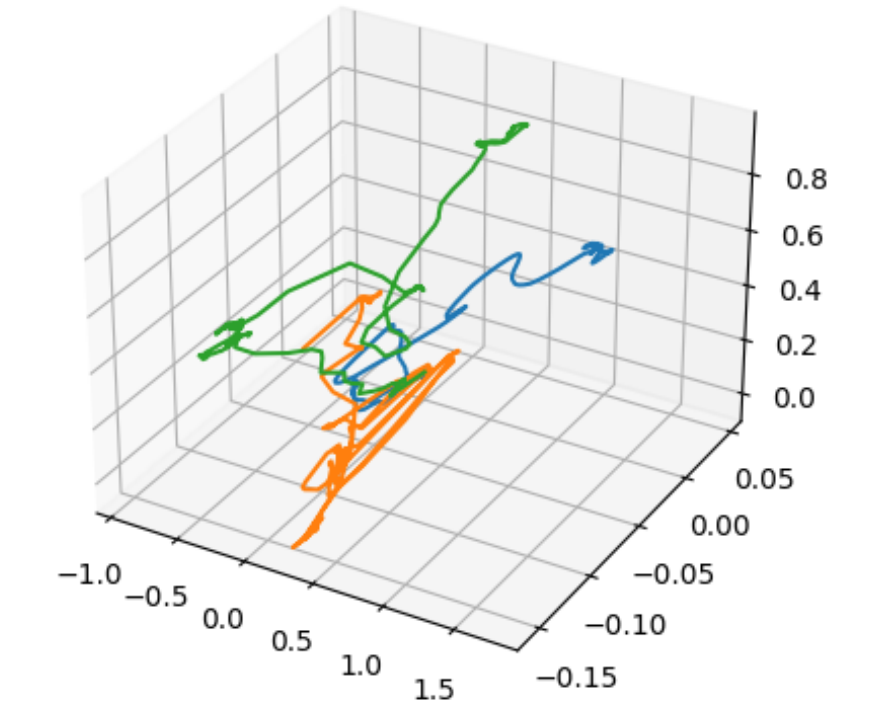


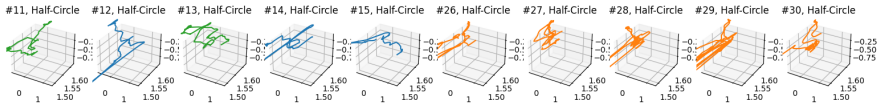


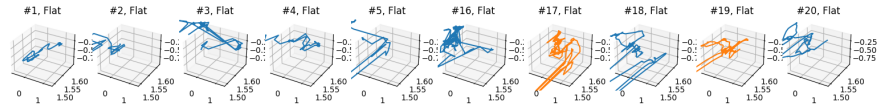


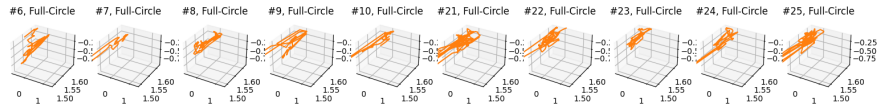


ANNEXE E : Soft K-Mean avec 1 cluster par layout

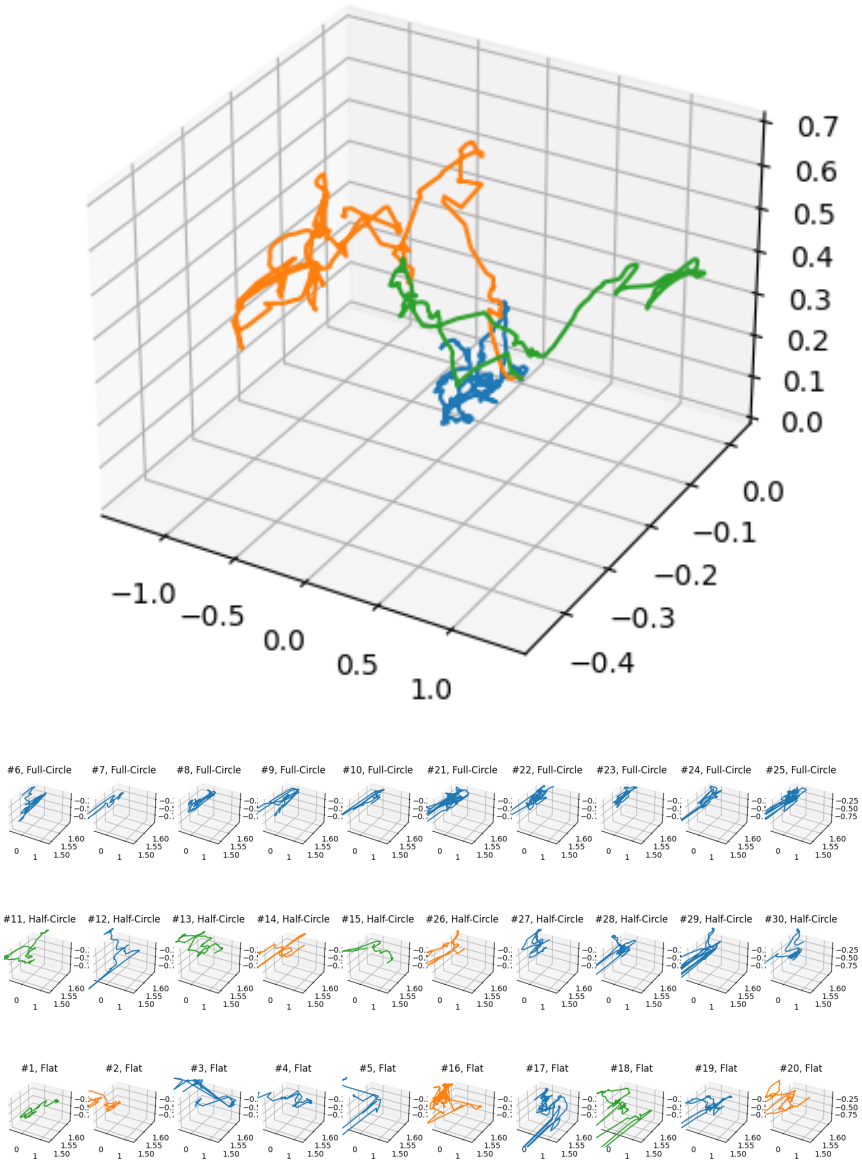




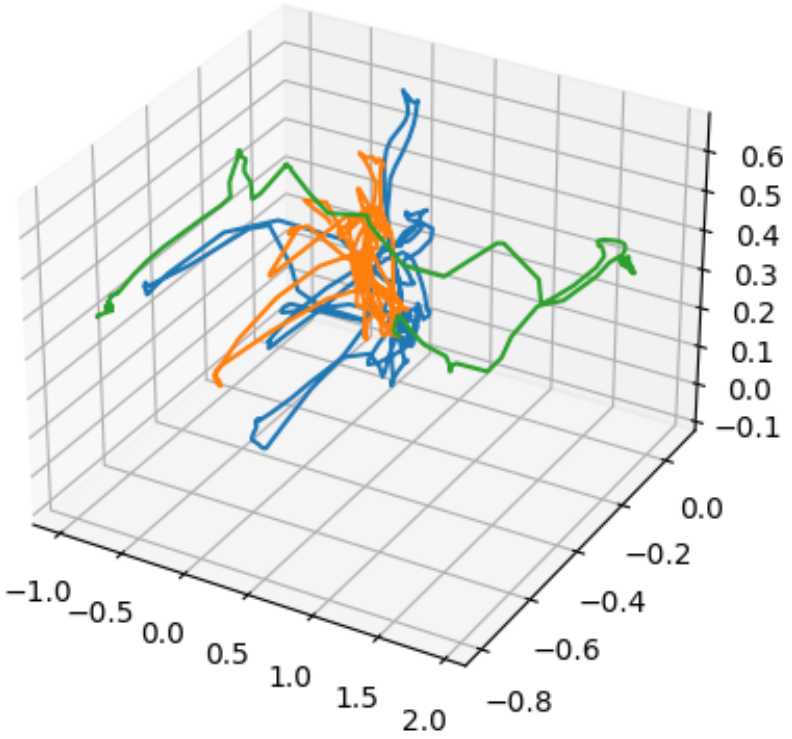


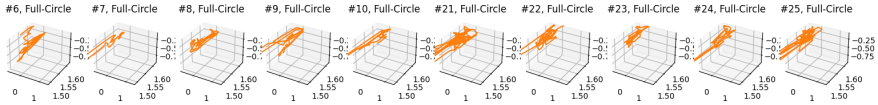


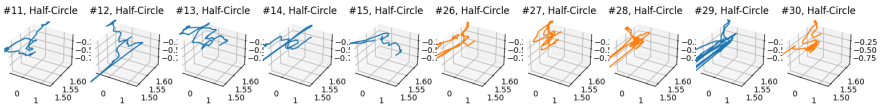
ANNEXE F : K-Mean avec 1 cluster aléatoires par layout

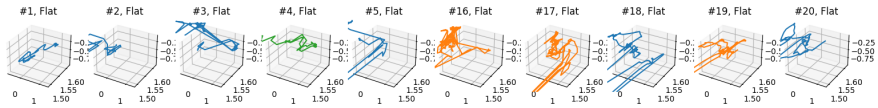


ANNEXE G : Soft K-Mean avec 1 cluster aléatoires par layout

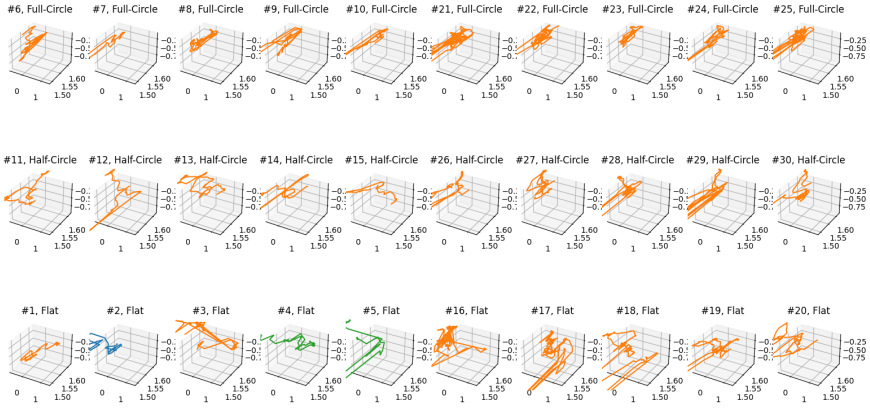




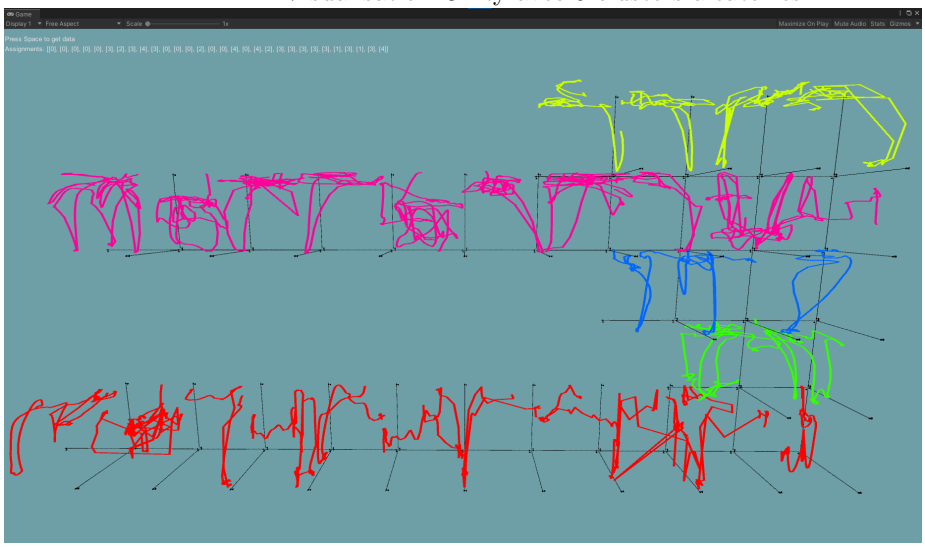




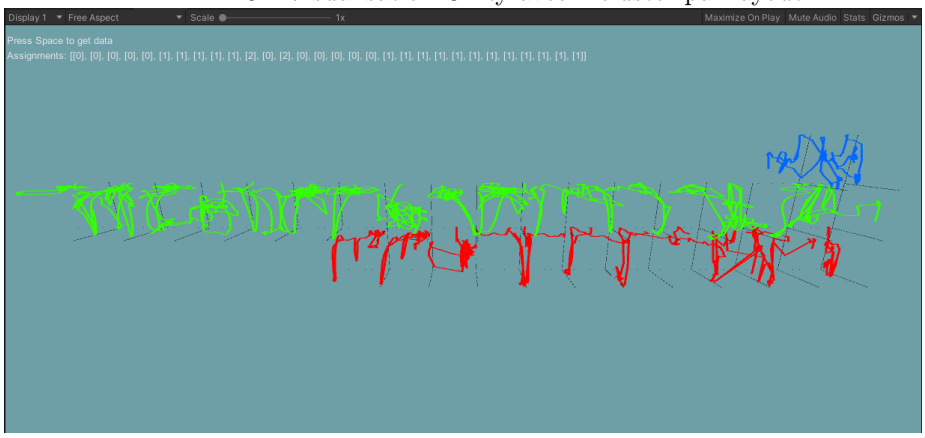
ANNEXE H : K-Mean d'OpenCV avec 3 clusters aléatoires



ANNEXE I : Visualisation Unity avec 5 clusters aléatoires



ANNEXE J : Visualisation Unity avec 1 cluster par layout



References

[1] Jiazhou Liu, Arnaud Prouzeau, Barrett Ens and Tim Dwyer. Design and Evaluation of Interactive Small Multiples Data Visualisation in Immersive Spaces. 2020.

[2] Cambridge University. An Example Inference Task: Clustering. 2003.

[3] Maxime Cordeil, Andrew Cunningham, Benjamin Lee. Immersive Analytics ToolKit IATK. 2019.