Rapport d'analyse des données ARE 2025

Lou Eouzan

3 avril 2025

Table des matières

1 Introduction **1**

2 Imports **1**

3 Ouverture du fichier et extraction des colonnes **1**

4 Obtention de l'allure de la montée et de la descente **1**

5 Tentative d'obtention de la vitesse **3**

6 Tentative d'obtention du temps d'escalade **3**

7 Calcul du ratio temps de mouvements sur temps total **4**

8 Tentative d'obtention de la hauteur escaladée **4**

9 Conclusion **5**

1 Introduction

Lors de l'avant dernière séance nous avons réalisé une prise vidéo que nous avons du analyser à l'aide du logiciel kinovéa. Via le logiciel,nous procèdons au tracking de la positions sur un axe x (horizontal) et un axe y (vertical) d'un point correspondant approximativement au centre de gravité tout du long de la vidéo. Sur ce rapport et cette vidéo là le haut du mur n'as pas été atteint contrairement au rapport précédent. L'objectif de ce rapport est d'abord d'analyser les mesures obtenues pour en déduires certaines données comme le rapport entre le temps de mouvement et le temps d'immobilité, la vitesse d'escalade, le temps d'escalade et la hauteur du mur mais également de comparer l'efficacité de ce mode de mesure avec celui utilisée lors du rapport 1 où nous avions mesuré l'accélérations linéaire sans g.

2 Imports

```
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
```

Ce code importe des bibliothèques python dont nous avons besoin. La bibliothèque matplotlib nous permet de produire des graphiques pour avoir une représentation visuelle des données. Enfin les bibliothèque numpy et pandas nous mettent à disposition des structures de données et des fonctions associées pour stocker et analyser nos mesures.

3 Ouverture du fichier et extraction des colonnes

```
data = pd.read_csv("esca.csv", sep=";", decimal=",")
time = data["Time"]
x = data["Trajectoire 1/0/X"]
y = data["Trajectoire 1/0/Y"]
```

La première ligne importe les données dans une variable data, ensuite les lignes suivantes stockent chaque colonne du dataframe obtenu en lisant les données dans une variable correspondante. Les variables x et y stockent la position de notre point sur l'axe horizontal et vertical. Enfin la variable time, elle contient les secondes correspondant au moment où chaque valeur de position a été mesurée.

4 Obtention de l'allure de la montée et de la descente

```
i_max = np.where(y == np.max(y))[0][-1]
print(i_max)
plt.clf()
plt.plot(x[:i_max], y[:i_max])
plt.xlabel("Position horizontale (en cm)")
plt.ylabel("Position verticale (en cm)")
plt.savefig("traj_montee.svg")
plt.clf()
plt.plot(x[i_max:], y[i_max:])
plt.xlabel("Position horizontale (en cm)")
plt.ylabel("Position verticale (en cm)")
plt.savefig("traj_descente.svg")
plt.clf()
```

```
3897
```

La première ligne renvoie l'index pour lequel la positions sur l'axe vertical est la plus haute. Cette position la plus haute correspond logiquement au haut du mur et donc à l'index de la fin de la montée. En l'occurence, l'index contenant la position la plus haute est l'index 3897. Pour afficher la trajectoire lors de la montée et la descente on veut afficher la position sur l'axe x et l'axe y respectivement de l'index 0 à l'index 3897 et de l'index 3898 à la fin du tableau. C'est ce dont ce charge les lignes suivantes.

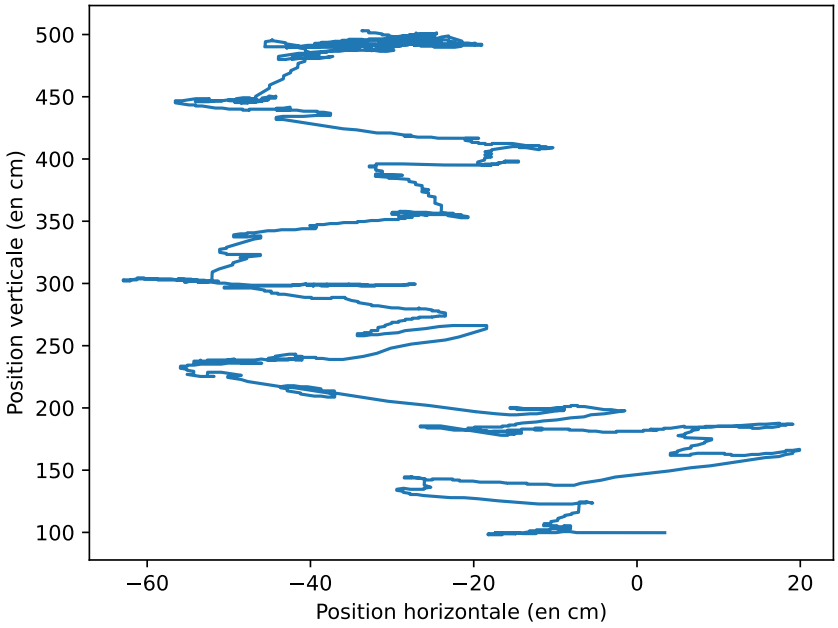

Fig. 1 : Tentative pour déterminer l'allure de la montée.

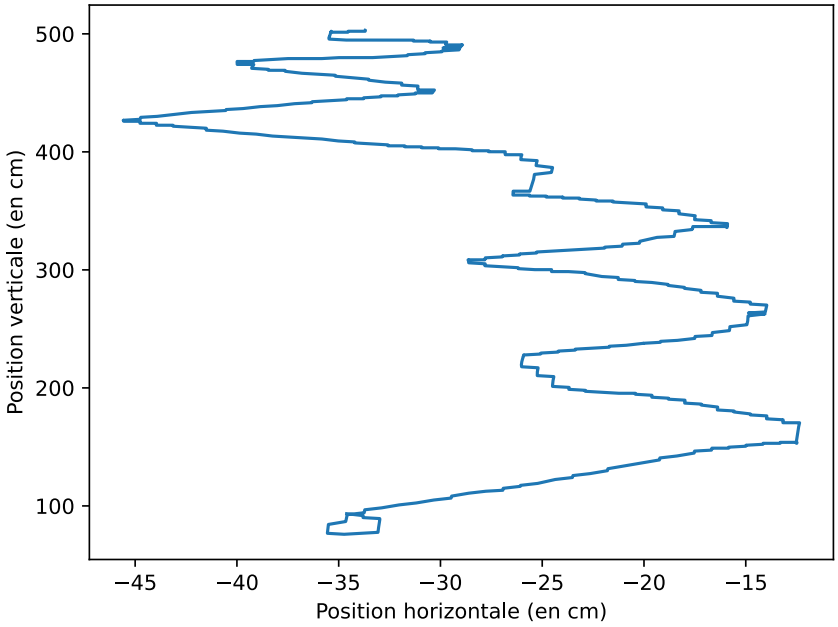

Fig. 2 : Tentative pour déterminer l'allure de la descente.

5 Tentative d'obtention de la vitesse

```
i = 0
vitesses_par_secondes = []
```

```
time_v= []
while i <= i_max-30:
    time_a, x_a, y_a = data.iloc[i]
    time_b, x_b, y_b = data.iloc[i+30]
    dist_en_cm = np.sqrt((x_b-x_a)**2+(y_b-y_a)**2)
    vitesses_par_secondes.append(dist_en_cm/(time_b-time_a)) # vitesse en cm/s
    time_v.append(time.iloc[i+30])
    i+=30

plt.plot(time_v, np.array(vitesses_par_secondes)*0.036)
plt.xlabel("Temps (en secondes)")
plt.ylabel("Vitesse (en km/h)")
plt.savefig("speed.svg")

print(np.mean(np.array(vitesses_par_secondes))*0.036) # vitesse moyenne en km/h
```

```
0.2571500349677964
```

Pour connaitre la vitesse en kilomètres par heure, il est nécesssaire de convertir les distances en kilomètre et le temps en heure, pour celà a été pris la mesure de la distance de ma taille en pixel sur la vidéo et le segment correspondant à été qualibré comme équivalent à 165 cm, kinovea c'est donc chargé de convertir lui même les distances en pixel en cm. Le temps quand à lui est en secondes. La différence de temps entre chaque mesures est, d'après une simple lecture des premières données d'environ 0,033 secondes, il faut environ 30 mesures $(1/0,033)$ pour avancer d'une seconde. Les positions dont nous mesurons la distance correspondent donc à celle de la mesure x et de la mesure $x+30$. Il suffit ensuite de faire une moyenne des vitesses mesurées lors de la montée en centimètres par secondes. Nous tentons en ensuite de convertir la vitesse en km/h, $1km = 100000cm$ donc on commence par diviser par 100 000, par ailleurs 1 heure = 3600 secondes donc on multiplie par 3600, on a donc $1cm/s = (1/100000) * 3600 = 0,036km/h$, il suffit donc de multiplier les cm/s par 0,036 pour obtenir la vitesse en km/h. La vitesse obtenue est donc d'environ 0,26 km/h. Dans la mesure où, sur la vidéo nous pouvons voir que je suis assez lente et que par ailleurs je fais beaucoup de pause et j'hésite à monter, cette vitesse moyenne, tout comme celle du rapport précédent, pourrait être cohérente. Elle l'est en tout cas plus que celle trouvée lors du premier rapport qui mesurait jusqu'à 10 000 m/s ($= 10000 * 3,6 = 36000km/h$ (on divise par 1000 pour que ce soit en km et on multiplie par 3600 pour que ce soit en heure ce qui revient à faire $x * (3600/1000) = x * 3,6$)) ce qui était totalement invraissemblable.

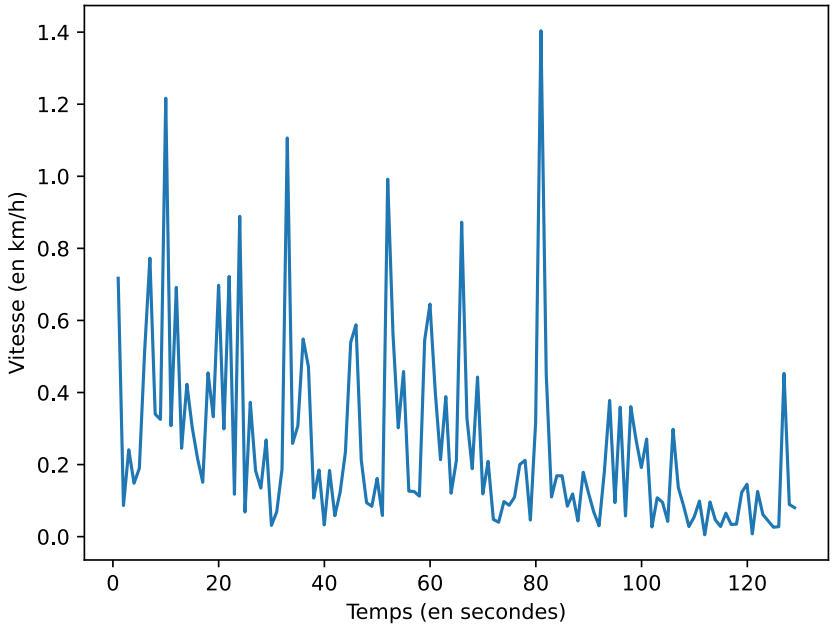

Fig. 3 : Vitesse mesurée au cours du temps lors de la montée.

6 Tentative d'obtention du temps d'escalade

```
print(time[i_max])
```

```
129.842
```

Pour mesurer le temps d'escalade il suffit tout simplement de regarder la valeur à l'index 3897 du tableau time correspondant au temps passé depuis le début quand nous avons atteint le point le plus haut sur l'axe vertical, c'est à dire le haut du mur. Que ce soit pour ce rapport, le rapport précédent ou le premier rapport, la mesure du temps était cohérente.

7 Calcul du ratio temps de mouvements sur temps total

```
def isclose(a, b, trsh):
    return abs(a-b)<trsh
trsh = 1
index_deb = time[(isclose(time, 38.62, trsh))]
index_fin = time[(isclose(time, 44.95, trsh))]
while len(index_deb) > 1:
    trsh/=10
    index_deb = time[(isclose(time, 38.62, trsh))]
trsh = 1
while len(index_fin) > 1:
    trsh/=10
    index_fin = time[(isclose(time, 44.95, trsh))]
index_deb = index_deb.index[0]
index_fin = index_fin.index[0]
print(index_deb, index_fin)
```

```
1159 1349
```

Pour trouver les temps d'immobilités, nous avons identifiés visuellement entre 38.62 secondes et 44.95 secondes un temps d'immobilité. Nous allons donc chercher les index correspondant le plus à ces timecode puis mesurer la distance moyenne considérée comme parcourue sur cet interval et considérer ensuite que toute distance parcourue supérieure à cette distance moyenne correspond à du mouvement. C'est cette recherche dont se charge ce code en retrecissant le nombre d'élément autour de ces timecodes jusqu'à ne trouver plus que l'élément le plus proche.

```
i = 0
dist_imo = []
while i < len(data[index_deb:(index_fin+1)])-30:
    time_a, x_a, y_a = data.iloc[i]
    time_b, x_b, y_b = data.iloc[i+30]
    dist_imo.append((np.sqrt((x_b-x_a)**2+(y_b-y_a)**2)))
    i+=30
trsh = np.mean(np.array(dist_imo))
print(trsh)
```

```
8.770730696971876
```

Ce code ci se charge de calculer la distance moyenne parcouru sur cet interval pour pouvoir considéré que toute période où la distance parcouru est supérieure ou égale à celle ci sera considérée comme une période de mouvement.

```
i = 0
mouvement = []
while i <= (len(time)-30):
    time_a, x_a, y_a = data.iloc[i]
    time_b, x_b, y_b = data.iloc[i+30]
    dist_en_cm = (np.sqrt((x_b-x_a)**2+(y_b-y_a)**2))
    if dist_en_cm > trsh:
        mouvement.append(True)
    else:
        mouvement.append(False)
    i+=30

mouvement_def = []
for j in mouvement:
    for k in range(30):
        mouvement_def.append(j)

if i < len(time):
    diff = len(time)-i
    for _ in range(diff):
        mouvement_def.append(mouvement_def[-1])

print(f"{round(((len(time[(np.array(mouvement_def))])*0.033)-0.033)/(time[3432])*100, 2)} %")
```

```
46.49 %
```

Pour chaque intervalle de 1 seconde environ où on considère qu'il y a du mouvement, on ajoute True dans le tableau mouvement pour l'indiquer sinon on ajoute false. Ensuite, nous créons un 2e tableau mais où chaque élément est présent 30 fois pour correspondre aux 30 index sautés pour faire nos mesures sur un intervalle de 1 seconde. Enfin on vérifie que le tableau définitif contient bien le même nombre d'éléments que **time** et si ce n'est pas le cas, on ajoute le dernier élément du tableau. C'est nécessaire car soit un tableau numpy quelconque arr et un tableau de booléen,

arr[(cond)] ne renvoie que les éléments de arr pour lequel cond est égal à True, en l'occurence correspondant aux moments où on est en mouvement. Chaque éléments de ce tableau correspondants à 0,033 secondes on multiplie la taille du tableau obtenu par 0,033 puis on soustrait une fois 0,033 puisqu'au premier élément du tableau 0 secondes se sont passées. Pour le temps total on prend le temps à l'index max. On fait ensuite le rapport entre les 2 chiffres obtenus puis on multiplie par 100 pour obtenir le pourcentage. Nous obtenons donc 46.49%, ça parait cohérent. Dans le dernier rapport avait été mesuré 33.43% avec à peu près le même temps de mouvement mais ce n'était pas cohérent par rapport au résultats du rapport 1. Nous en avions conclu que la méthode avec accéléromètre était donc plus efficace, mais cette fois ci en ayant plus d'expèrience avec le logiciel et donc ayant réalisé un meilleur suivi de la trajectoire de mon centre de gravité le résultat parait plus cohérent. Nous povons donc supposer que, soit les résultat sont plus cohérent avec expèrience, mais on peut aussi supposer qu'il y ai une marge d'erreur systématique qui rend trop aléatoir la cohérence de ce résultat.

8 Tentative d'obtention de la hauteur escaladée

```
print((np.mean(np.array(vitesses_par_secondes))/100)*(time[i_max]))
```

```
9.27468745563573
```

Nous savons que $vitesse = distance/temps$, ça signifie donc que $distance = vitesse * temps$. Nous avons la vitesse en cm/s, il suffit de la diviser par 100 pour obtenir la vitesse en mètre par secondes. Nous multiplions la vitesse par le temps total d'escalade et on obtient un résultat d'environ 9,2 mètres. Ce résultats parait peu réaliste dans la mesure où dans le rapport précédent avait été mesuré 9.6 mètres pour la hauteur du mur, que je n'ai sur ce coup là pas escaladé en entier. Nous pouvons supposer une imprecision dans le tracking.

9 Conclusion

Dans ce rapport, nous avons mesuré la positions de mon centre de gravité dans un plan. De cette position et de son évolutions à travers le temps nous avons obtenus la vitesse, le temps d'escalade, le ratio temps de mouvements sur temps total et la hauteur du mur. l'imprécision potentielle du tracking supposée dans la section d'obtention de la hauteur d'escalade contredirait les conclusions de la section qui la précéde et remettrai en question la cohérence des résultat obtenus dans la section en question de ce rapport et son équivalent dans le rapport 1.