



Estimation de l'orientation d'un ombilical sous-marin à partir de centrales inertielles

Encadrantes :

DUNE Claire
DRUPT Juliette

Etudiants :

MINATCHY Lana
MABILOTTE Grégory

SYSMER 3A
Année universitaire 2022-2023

Sommaire

1 Contexte et objectif	3
2 Définitions	4
2.1 Théorie	4
2.1.1 Angles d'Euler	4
2.1.2 Quaternions	5
2.2 Matériel	6
2.2.1 Centrale inertielle <i>Phidget</i>	6
2.2.2 Qualisys	7
3 Prise en main du Phidget / Tests préliminaires	8
3.1 Extraction des données	8
3.2 Tests de l'extraction des données, du comportement du Phidget et résolution des problèmes	9
3.2.1 Expériences sur le roulis	9
3.2.2 Expériences sur le tangage	10
3.2.3 Expériences sur le lacet	11
4 Test physique pour la validation des données et de la robustesse du Phidget avec le système Qualisys	13
4.1 Installation et calibrage	14
4.2 Modalités d'expérience	15
5 Analyse des données	16
5.1 Problèmes et résolution	16
5.1.1 Synchronisation des données	16
5.1.2 Fréquences de mesure différentes	17
5.1.3 Repères différents	18
5.2 Calcul des matrices de rotation	19
5.3 Comparaison des matrices	20
5.4 Visualisation des rotations	20
6 Conclusion	22

7 Annexes	23
7.1 Code Python : Exécution des Phidgets et écriture des données en temps réel : <i>Phidget.py</i>	23
7.2 Code Python : Extraction des données Euler depuis les fichiers IMU : <i>Euler-File_IMU.py</i>	26
7.3 Code Python : Extraction des données Quaternions depuis les fichiers IMU : <i>QuaternionsFile_IMU.py</i>	28
7.4 Code Python : Tracé des courbes des angles d'Euler de l'IMU : <i>plot.py</i>	29
7.5 Code Python : Calcul des données Quaternions Qualisys depuis les données Euler Qualisys : <i>QuaternionsFile_Qualisys.py</i>	31
7.6 Code Python : Extraction des données des matrices de rotations depuis les fichiers Qualisys : <i>RotFile_Qualisys.py</i>	34
7.7 Code MATLAB : Matrice de rotation	36
7.8 Code MATLAB : comparaison matrices de rotation	38
7.9 Code MATLAB : création des vidéos GIF	39

1 Contexte et objectif

L'utilisation de ROV ou AUV permet d'améliorer grandement l'exploration des milieux sous-marins. En effet, ils permettent d'explorer des endroits peu ou pas accessibles à l'homme et très contraignants. De plus, ils permettent de ramener de ces endroits de nombreuses informations grâce à la multitude de capteurs qu'ils peuvent emporter avec eux.

Les ROV utilisés pour ce projet sont de type BlueROV. Un ROV est relié à un ordinateur à la surface avec un câble, appelé ombilical, qui permet de transmettre les données relevées par le robot à l'utilisateur. Cet ombilical entrave parfois les mouvements du robot à cause de la traction qu'il crée, c'est pourquoi le laboratoire COSMER a décidé d'étudier une cordée de robots, c'est-à-dire plusieurs ROV relié entre eux qui suivent un ROV de tête téléopéré. Comme chacun des câbles créé une traction, il est important de connaître leurs formes pour anticiper l'impact que la traction aura sur les mouvements des autres ROV.

L'objectif de ce projet est de déterminer l'orientation d'un ombilical sous-marin lors de son utilisation avec un ROV. On utilise pour cela une centrale inertie qui nous permettra d'obtenir une représentation d'Euler (roulis, tangage et lacet) ou en quaternion de l'inclinaison de l'ombilical.

Nous allons donc dans un premier temps expliquer les différents concepts théoriques et le matériel utilisés (Euler, quaternions, centrale inertie Phidget, système Qualisys). Puis nous présenterons la prise en main de la centrale inertie et les tests préliminaires réalisés. Enfin, nous caractériserons les expériences de validation réalisées avec le système Qualisys et nous analyserons les données obtenues afin de réussir à déterminer la forme d'un ombilical.

2 Définitions

Dans cette première partie, nous présenterons les différentes théories relatives aux rotations dans l'espace utilisées dans ce projet. Puis, nous présenterons le matériel utilisé pour réaliser les expériences et obtenir nos résultats.

2.1 Théorie

Les deux représentations de rotation que nous avons utilisées sont : les angles d'Euler et les quaternions.

2.1.1 Angles d'Euler

Les angles d'Euler sont un système de coordonnées qui permettent de décrire l'orientation d'un objet dans l'espace à l'aide de trois angles. Ces angles correspondent au roulis (rotation autour de l'axe x), au tangage (rotation autour de l'axe y) et au lacet (rotation autour de l'axe z). Ces angles sont en général représentés respectivement par les angles ϕ , θ et ψ . Ces angles sont illustrés sur les images ci-dessous.

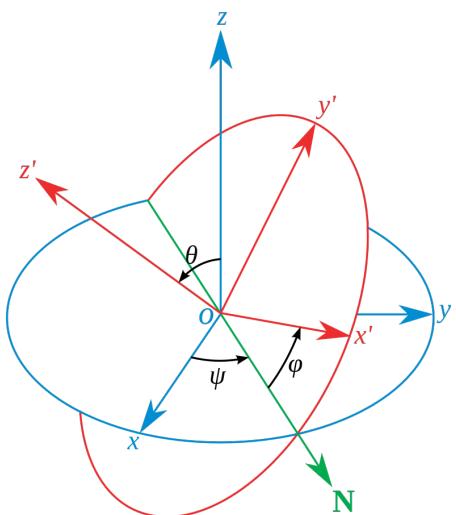


FIGURE 1 – Représentation des angles d'Euler

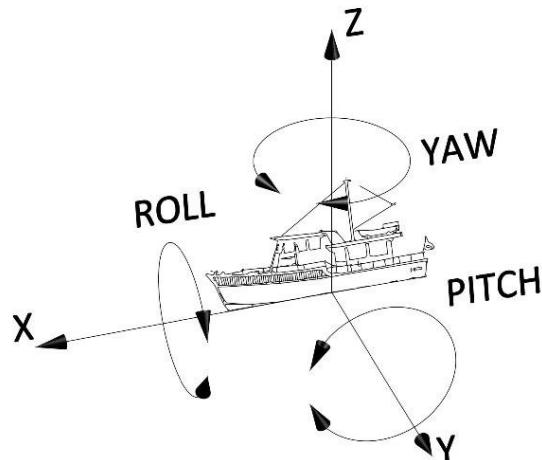


FIGURE 2 – Rotations de roulis (roll), tangage (pitch) et lacet (yaw)

Ces angles sont très utilisés dans tous les domaines où il est important et nécessaire de décrire l'orientation d'un objet dans l'espace comme la navigation ou la robotique. Cependant, ils peuvent aussi être à l'origine de problèmes de singularité ou de discontinuité. En effet, une rotation trop importante autour de certains axes peut amener physiquement le système à perdre des degrés de liberté. Ce problème est communément connu sous le nom de Gimbal Lock. Pour éviter les problèmes de ce type là, d'autres systèmes de coordonnées tels que les quaternions sont souvent choisis.

2.1.2 Quaternions

Les quaternions sont une extension des nombres complexes, avec non plus une unique partie imaginaire mais trois. On peut alors écrire un quaternion sous la forme :

$$q = a + b.i + c.j + d.k \quad (1)$$

Où, a, b, c et d sont des scalaires et i, j et k sont des nombres imaginaires.

Il s'agit donc d'un système à 4 dimensions. Les quaternions sont utilisés pour représenter des rotations en 3 dimensions. On peut penser les quaternions comme une division de deux vecteurs ou un scalaire associé à un vecteur à 3 dimensions.

Ils obéissent au tableau de multiplication suivant.

	1	i	j	k
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

FIGURE 3 – Règle de multiplication des quaternions (gauche * haut)

On remarque que, de manière équivalente aux nombres complexes, on a : $i^2 = j^2 = k^2 = ijk = -1$. Par ailleurs, pour les quaternions, l'ordre des multiplication est importante. En effet, $ij = k$ et $ji = -k$.

Il est possible de convertir des angles d'Euler en quaternions en utilisant la formule (2), où R est l'angle de roulis, P l'angle de tangage et Y l'angle de lacet.

$$(\cos \frac{R}{2} + i \cdot \sin \frac{R}{2}) * (\cos \frac{P}{2} + j \cdot \sin \frac{P}{2}) * (\cos \frac{Y}{2} + k \cdot \sin \frac{Y}{2}) \quad (2)$$

En développant et en utilisant les règles du multiplication figure 3, on obtient alors q comme défini équation 1, avec a scalaire, b associé à i , c associé à j , d associé à k et :

$$\left\{ \begin{array}{l} a = \cos \frac{R}{2} \cos \frac{P}{2} \cos \frac{Y}{2} - \sin \frac{R}{2} \sin \frac{P}{2} \sin \frac{Y}{2} \\ b = \sin \frac{R}{2} \cos \frac{P}{2} \cos \frac{Y}{2} + \cos \frac{R}{2} \sin \frac{P}{2} \sin \frac{Y}{2} \\ c = \cos \frac{R}{2} \sin \frac{P}{2} \cos \frac{Y}{2} - \sin \frac{R}{2} \cos \frac{P}{2} \sin \frac{Y}{2} \\ d = \cos \frac{R}{2} \cos \frac{P}{2} \sin \frac{Y}{2} + \sin \frac{R}{2} \sin \frac{P}{2} \cos \frac{Y}{2} \end{array} \right. \quad (3)$$

Les quaternions, une fois définis pour un repère, sont uniques à l'inverse des angles d'Euler qui dépendent de la convention dans laquelle ils sont déterminés. Ainsi, si on reconvertis des quaternions en angles d'Euler, il est très probable de ne jamais retrouver les angles d'origine. C'est pourquoi les quaternions sont plus souvent utilisés lorsqu'il faut comparer des rotations entre des repères différents. De plus les quaternions permettent de s'affranchir du Gimbal Lock qui est une contrainte majeure de l'utilisation des angles d'Euler.

2.2 Matériel

Pour notre projet, nous avons utilisé des centrales inertielles de type *Phidget* pour relever les données. Nous avons aussi utilisé un système Qualisys pour valider les données obtenues par la centrale惯性

2.2.1 Centrale inertuelle *Phidget*

Une centrale inertuelle (Inertial Measurement Unit, IMU) permet de déterminer l'orientation dans l'espace d'un objet en 3D. Elle fournit les angles de roulis, tangage, lacet et le cap d'un objet. Si elle est couplée avec un GNSS (Global navigation satellite system, exemple GPS), elle peut aussi fournir la position, la vitesse de l'objet. À l'inverse d'un GNSS, une centrale inertuelle ne nécessite aucune donnée extérieure afin de calculer l'orientation et le cap d'un objet.

Une centrale inertuelle basique est composée de 3 accéléromètres, 3 gyroscopes (un pour chaque axe x, y, z), et un calculateur (filtre de Kalman). Dans la majorité des IMU, on retrouve aussi 3 magnétomètres. Un accéléromètre mesure l'accélération du mouvement d'un objet selon un axe. Un gyroscope mesure la vitesse angulaire d'un objet autour d'un axe. Un magnétomètre mesure la direction et l'intensité du champ magnétique de la Terre pour calculer le cap magnétique d'un objet.

Pour notre projet, nous avons travaillé sur les IMU de type *PhidgetSpatial Precision 3/3/3 MOT0109_0*, développé par l'entreprise Phidgets, présenté figure 4.



FIGURE 4 – Centrale inertuelle *Phidget* utilisée

L'orientation du repère du Phidget est dessinée sur le boîtier. L'axe 0 est l'axe de rotation correspondant au roulis, l'axe 1 est celui correspondant au tangage et l'axe 2 correspond à

l'angle de lacet. Cet axe est orienté vers le haut. Le repère de référence monde permettant de calculer les valeurs du roulis, tangage et lacet de notre Phidget est orienté selon la convention suivante : l'axe x est orienté vers le Nord magnétique, l'axe y est orienté vers l'Ouest et l'axe z vers le haut. Les valeurs retournées par le Phidget pour le roulis, le tangage et le lacet sont les angles qui séparent les deux repères si nous fixons les deux à la même origine.

2.2.2 Qualisys

Le système Qualisys permet de capturer, suivre et analyser des mouvements en 3D. Il est composé de deux parties, une partie physique (les caméras) et la partie logiciel (Qualisys Track Manager (QTM)).

- Caméras

Les caméras du Qualisys, présentées figure 5, permettent de capter les mouvements en 3D d'objets présents dans leurs champs de vision.



FIGURE 5 – Caméra du Qualisys



FIGURE 6 – Marqueur

Pour capter les différents mouvements, on utilise des marqueurs (figure 6) qui reflètent les ondes ensuite captées par les caméras.

Les données en temps réel sont envoyées à l'ordinateur sur le logiciel QTM depuis lequel il est possible de les récupérer pour ensuite les analyser.

- Logiciel QTM

Le logiciel QTM (Qualisys Track Manager) est celui utilisé pour recueillir les données des caméras et les exporter sous différentes formes (*tsv, mat*) afin de les analyser.

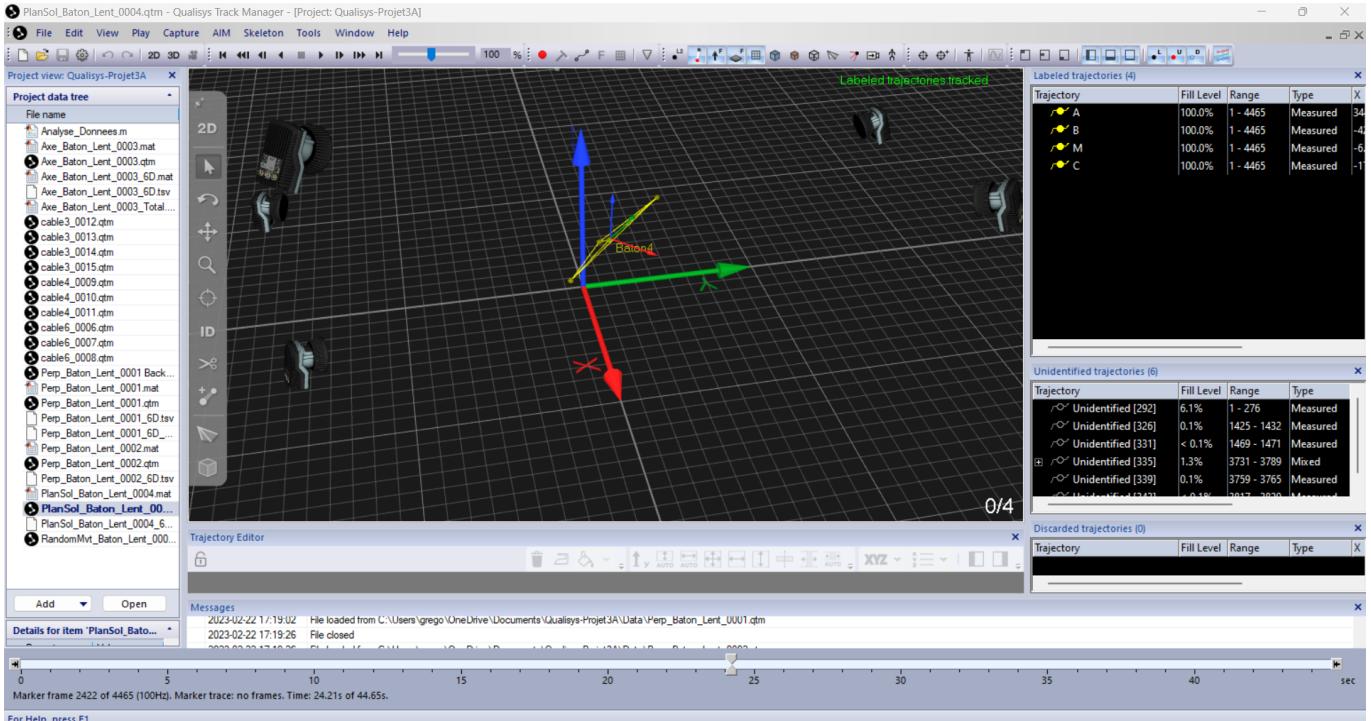


FIGURE 7 – Interface QTM

Dans cette fenêtre, on a, à gauche, les fichiers *qtm* relatifs aux différentes acquisitions et les fichiers *tsv* et *mat* exportés. Au milieu, on a le retour vidéo de l'acquisition que l'on peut rejouer et mettre en pause. On peut aussi ajouter un solid body pour obtenir les angles d'Euler des différents points (équivalent aux marqueurs). Enfin à droite on a la liste des points référencés et leur pourcentage d'acquisition, c'est-à-dire le pourcentage de mesures qui réussit à être effectuées pour chaque point au cours de la manipulation (100% si le système a bien réussi à tracer le marqueur tout le long de l'acquisition).

3 Prise en main du Phidget / Tests préliminaires

Nous avons commencé par prendre en main le Phidget que nous avions à notre disposition. Pour cela, nous avons regardé sa documentation et les différentes fonctions qui étaient fournies avec la bibliothèque *Phidget 2022*. La documentation nous a permis de comprendre comment le système et le Phidget interagissent via des channels. Elle nous a aussi permis de trouver les différentes fonctions qui nous ont permis d'extraire les données en temps réel du Phidget, notamment les angles d'Euler et les quaternions.

3.1 Extraction des données

Afin d'extraire les données du Phidget en temps réel, nous avons utilisé l'environnement Linux/Ubuntu et le langage de programmation *Python* car celui-ci est moins contraignant que les autres langages de programmation disponibles pour interagir avec le Phidget. Nous avons commencé par essayer de récupérer les données d'un seul Phidget.

Pour cela, nous avons dû prendre en main la nouvelle bibliothèque *Spatial API*. Celle-ci fournit deux fonctions qui nous permettent de récupérer les données en temps réel des angles d'Euler et des quaternions : les fonctions *getEulerAngles()* et *getQuaternion()*. Pour récupérer les données en temps réel, nous avons écrit simultanément dans un fichier csv les données fournies par ces deux fonctions.

Afin de récupérer les données de deux Phidgets simultanément, nous avons dû modifier légèrement le code d'extraction des données. Nous avons associé chaque channel au numéro de série du Phidget pour lequel il devait récupérer les données. De plus, nous avons rajouter des channels pour l'extraction des données du second Phidget.

Le code effectuant cette extraction est donné en annexe 7.1.

Après avoir réussi à extraire des données, nous avons réalisé des tests pour les valider et pouvoir affirmer que le Phidget retourne bien les bons angles.

3.2 Tests de l'extraction des données, du comportement du Phidget et résolution des problèmes

Afin de tester notre programme Python d'extraction (disponible en annexe 7.1) et de comparer les données affichées par le Phidget et la réalité nous avons réalisé plusieurs séries de tests. Nous avons fait des tests pour chacune des valeurs des angles d'Euler : le roulis (roll), le tangage (pitch) et le lacet (yaw). Nous avons pour cela récupéré un cercle trigonométrique avec des valeurs d'angles remarquables : 30°, 45°, 60° etc., comme le montre la figure ci-dessous. Nous avons utiliser ce cercle pour fixer le Phidget à certaines valeurs et voir s'il nous retourne bien les mêmes valeurs que le cercle trigonométrique.

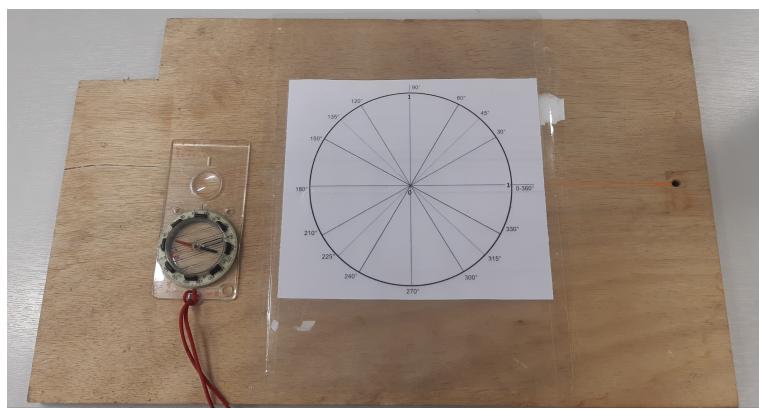


FIGURE 8 – Cercle trigonométrique et boussole utilisés

3.2.1 Expériences sur le roulis

La première expérience que nous avons faite a été de faire évoluer le roulis selon les différents angles du cercle trigonométrique à notre disposition. Entre chaque angle, nous attendions quelques secondes pour être sûrs que le Phidget converge vers les bonnes valeurs. L'analyse

des graphiques ci-dessous nous permet de conclure sur la véracité des informations données par le capteur pour le roulis. Le code permettant de tracé les courbes à partir des données de l'IMU est disponible en annexe 7.4

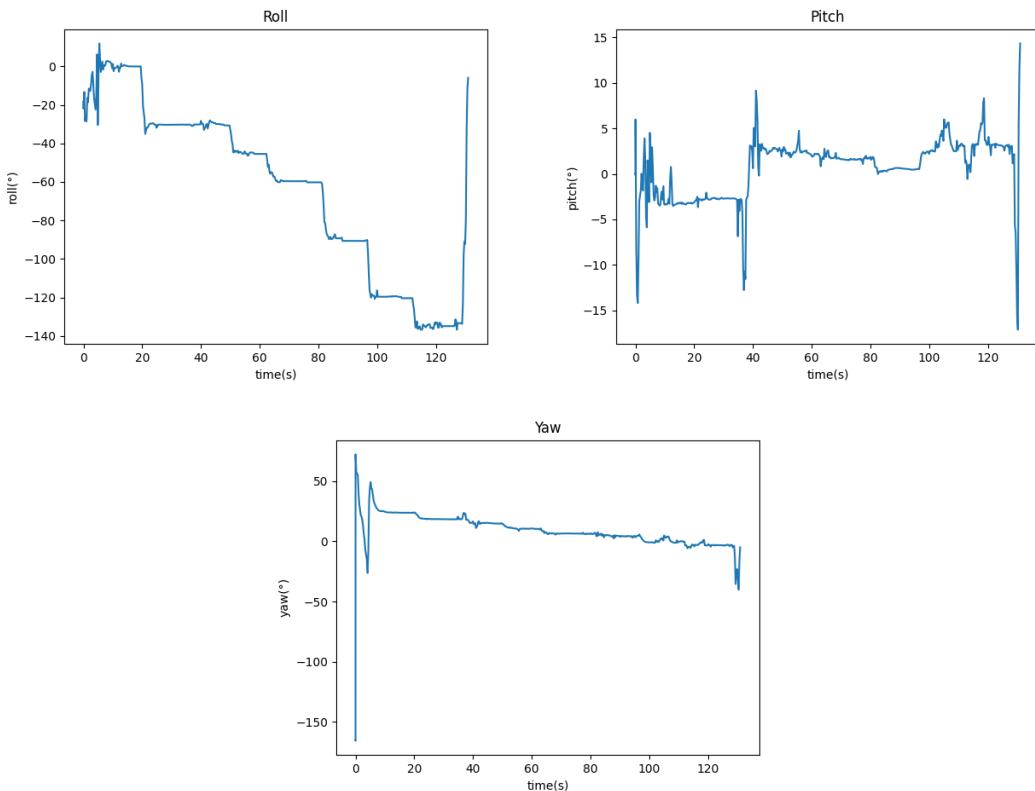


FIGURE 9 - Évolution des différents angles lors de l'expérience pour vérifier la cohérence du roulis par rapport à la réalité

Nous pouvons remarqué ici que pour le roulis nous observons bien les différents paliers correspondants au moment où nous avons attendu avant de changer d'angle. De plus, les valeurs de roulis affichées correspondent bien à celles de la réalité et du cercle trigonométrique. Nous observons également que le tangage et le lacet ne sont pas constants. Vu l'évolution de ces courbes, cette non constance est probablement due à une erreur de manipulation car les côtés du Phidget ne sont pas exactement plats et donc en faisant bouger le Phidget vers un nouvel angle de roulis, les angles de tangage et de lacet ont pu se décaler légèrement. Toutefois, les valeurs affichées par le Phidget reste satisfaisantes par rapport à la réalité, même si il a besoin d'un petit temps pour converger vers la valeur finale.

3.2.2 Expériences sur le tangage

La seconde expérience que nous avons faite est sur l'angle de tangage. Nous avons également utilisé le cercle trigonométrique pour nous assurer que le Phidget nous donne bien les bonnes valeurs. Nous avons également attendu quelques secondes entre chaque angle pour que cela soit bien visible sur les graphiques ci-dessous. Le code permettant de tracé les courbes à partir des données de l'IMU est disponible en annexe 7.4

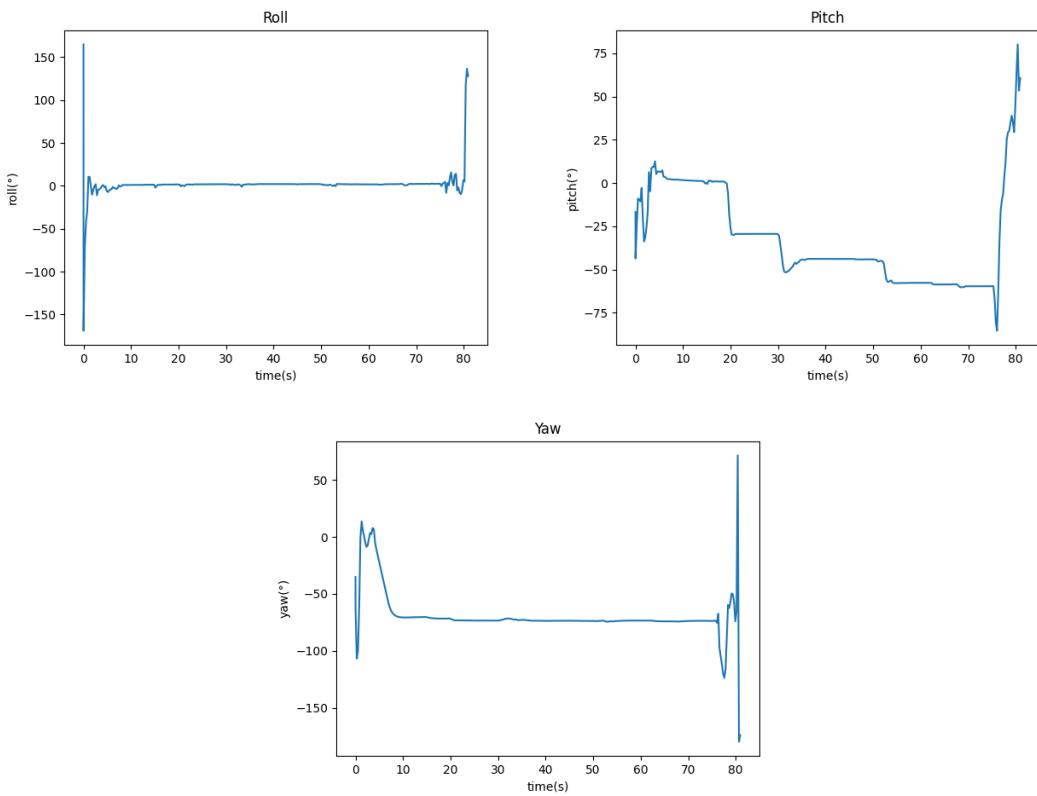


FIGURE 10 - Évolution des différents angles lors de l'expérience pour vérifier la cohérence du tangage par rapport à la réalité

Sur les graphiques ci-dessus, nous observons que l'angle de tangage suit bien les différents paliers que nous avons marqués lors de l'expérience. De plus, les valeurs des angles affichées correspondent à ceux du cercle trigonométrique. Toutefois, nous observons que lorsque nous arrivons avec des angles de tangage supérieur à 75° en valeur absolue, alors le Phidget se met à afficher des valeurs fausses pour les trois angles d'Euler. Ces valeurs fausses sont à cause d'un problème intrinsèque aux angles d'Euler : le blocage de Cardan ou *Gimbal Lock*. En effet, lorsque l'angle de tangage arrive aux alentours de 90° , alors deux axes de rotations commencent à se superposer. Le système perd alors un angle de liberté et il ne sait plus comment se situer dans l'espace, ce qui explique les valeurs fausses. Le seul moyen d'empêcher ce problème est de passer en quaternions ou de faire en sorte que le système n'aille jamais dans cette plage de données.

3.2.3 Expériences sur le lacet

Après avoir vérifié les angles de roulis et de tangage, nous avons vérifié la cohérence de l'angle de lacet par rapport à la réalité. Lors de nos premières expériences, nous avons laissé le magnétomètre activé, or la salle dans laquelle nous faisions les expériences est composée d'éléments métalliques, ce qui dérègle complètement le Phidget. Nous avons alors désactivé le magnétomètre avant de refaire des expériences. Pour s'assurer que le magnétomètre était bien désactivé, nous avons reproduit deux fois la même expérience pour le lacet, une dans la salle avec les

éléments métalliques et une à l'extérieur. Les graphiques des deux expériences sont affichés ci-dessous. Le code permettant de tracer les courbes à partir des données de l'IMU est disponible en annexe 7.4

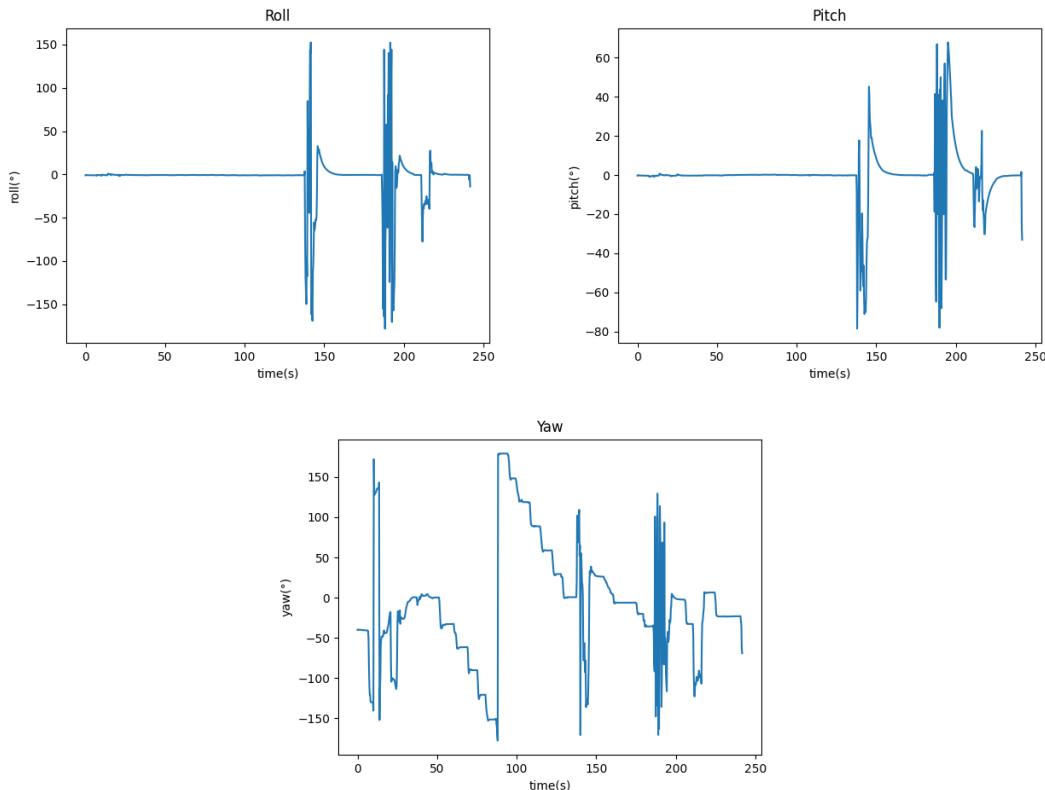


FIGURE 11 - Évolution des différents angles lors de l'expérience en intérieur avec le magnétomètre désactivé pour vérifier la cohérence du lacet par rapport à la réalité

Sur les figures ci-dessus, nous remarquons bien les paliers pour les angles de lacet et les valeurs affichées sont cohérentes avec celles du cercle trigonométrique. Nous voyons aussi que l'angle de lacet est compris entre -180° et 180° . Nous remarquons également que les angles de tangage et de roulis sont constants et égaux à zéro tout au long de l'expérience, ce qui est normal car nous n'avons pas fait évoluer ces deux angles. Nous avons également essayé de remuer le capteur dans tous les sens au milieu de l'expérience et de le replacer sur le même angle pour voir si les valeurs affichées étaient les mêmes. Nous pouvons voir que pour le tangage et le roulis, il n'y a pas de soucis et les valeurs restent à zéro. Cependant, pour l'angle de lacet, la valeur n'est pas tout à fait la même et le Phidget donne l'impression qu'il redéfinit un zéro magnétique. Nous n'avons pas trouvé comment corriger ce problème.

Les graphiques ci-dessous sont les résultats de l'expérience en extérieur.

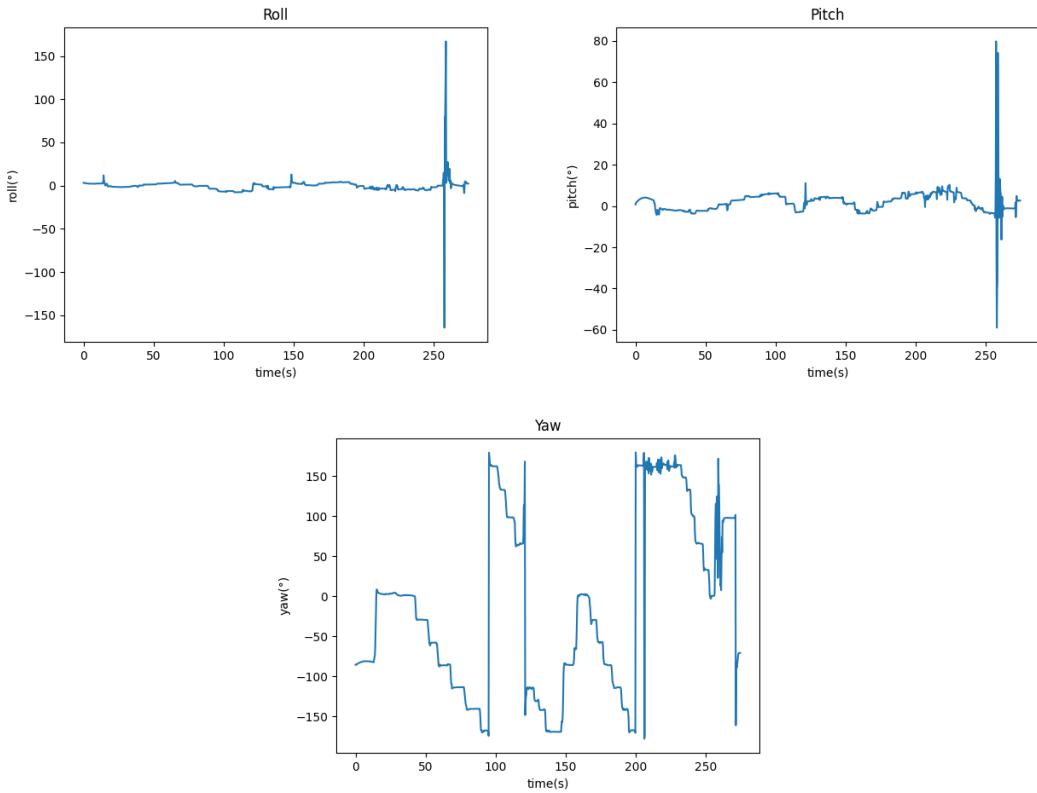


FIGURE 12 - Évolution des différents angles lors de l'expérience en extérieur avec le magnétomètre désactivé pour vérifier la cohérence du lacet par rapport à la réalité

Nous pouvons observer, comme pour l'expérience précédente, que les paliers sont présents pour l'angle de lacet et que les valeurs sont cohérentes avec le cercle trigonométrique. Nous observons également que les valeurs des angles de roulis et tangage ne sont pas constantes. Cela est dû au sol extérieur qui n'était pas forcément parfaitement droit. Nous remarquons aussi que le fait de secouer le Phidget dans tous les sens lui fait également redéfinir un zéro magnétique bien que nous étions en extérieur et qu'il n'y ait pas de nuisances magnétiques.

4 Test physique pour la validation des données et de la robustesse du Phidget avec le système Qualisys

Le but de cette expérience est de valider les données roll, pitch, yaw déterminées par la centrale inertie et récupérées par notre code Python.

Lors de ces manipulations, nous récupérons à la fois les angles d'Euler et les quaternions perçus par l'IMU grâce à notre code Python, ainsi que les angles d'Euler et les matrices de rotation calculés par le système Qualisys.

4.1 Installation et calibrage

Pour nos manipulations, nous avons utilisé 8 caméras Qualisys que nous avons placées en cercle et à l'intérieur duquel nous avons réalisé les différents mouvements de l'IMU. Les caméras sont reliées à un ordinateur sur lequel tourne le logiciel QTM afin de récupérer les données.



FIGURE 13 – Installation des caméras pour l'expérience

Les caméras doivent toutes être tournées vers la zone dans laquelle l'expérience sera réalisée. Sur la figure 13, cette zone se trouve juste au dessus du bloc rouge. De plus, il ne faut pas que les caméras soit toutes dans le même plan afin de pouvoir capter les marqueurs à chaque instant peu importe leurs positions. C'est pourquoi nous les avons placées à différentes hauteurs, comme on peut le voir sur la figure 13.

Nous avons aussi utilisé un bâton sur lequel nous avons attaché l'IMU à peu près au milieu. Afin de pouvoir capter les mouvements du bâton avec le système Qualisys, nous lui avons accroché 4 marqueurs : deux à ses extrémités (les points A et B), un au milieu (point M) et un sur l'IMU (point C), comme présenté figure 14.



FIGURE 14 – Dispositif pour l'expérience



FIGURE 15 – Mire de calibrage

Pour que le logiciel soit calibré correctement, il faut faire en sorte qu’au moins 4 caméras du Qualisys voit uniquement les 4 marqueurs de la mire. Pour aider le logiciel, nous pouvons déplacer les caméras si certains points sont en dehors du champ de vision d’une caméra ou enlever directement sur le logiciel certains points qui correspondent à des reflets d’objets de l’environnement autre que les marqueurs. Nous pouvons également jouer sur les contrastes de la caméra pour cacher ou faire apparaître des reflets/marqueurs au logiciel.

4.2 Modalités d’expérience

Une fois le logiciel calibré, nous avons pu commencé nos manipulations. Nous en avons réalisé 5 au total.

Le but des quatre premières étaient de faire tourner le bâton selon un seul axe de rotation (roll, pitch et yaw) afin d’étudier les rotations indépendamment les unes des autres. Nous avons recommencé une deuxième fois la manipulation du roll. Ces quatre manipulations sont nommées respectivement *Perpendiculaire bâton 1 et 2*, *Axe bâton* et *Plan sol*.

Lors de la dernière manipulation, appelée *Random*, nous avons réalisé des rotations aléatoires en tournant selon plusieurs axes simultanément. Cette manipulation n’a pas pu être exploitée car certains marqueurs sortaient du cadre et n’étaient donc plus pris en compte par le système Qualisys pendant des périodes de temps trop importantes pour être négligées.

5 Analyse des données

Après avoir récupéré les données de l'IMU et du Qualisys, nous avons été confrontés à plusieurs problèmes pour l'analyse des données. En effet, les deux systèmes n'ayant pas été lancé au même moment, un problème de synchronisation des données est apparu. De plus, l'IMU et le Qualisys avaient des fréquences d'acquisition différentes, c'est-à-dire qu'ils n'ont pas récupéré autant de valeurs l'un que l'autre. Enfin, le plus gros problème rencontré est que les repères de l'IMU et du Qualisys sont différents. Ainsi, afin de pouvoir comparer leurs données, il faut réussir à exprimer les rotations dans un seul repère ou trouver une solution pour contrer ce problème.

Les données récupérées de l'IMU sont :

- timestamp (date à laquelle la ligne de données a été récupérée)
- angles d'Euler : roll, pitch, yaw
- coefficients a, b, c, d du quaternion

Les données récupérées du Qualisys sont :

- numéro de ligne
- timestamp
- coordonnées x, y, z
- angles d'Euler : roll, pitch, yaw
- coefficients de la matrice de rotation

5.1 Problèmes et résolution

5.1.1 Synchronisation des données

Comme nous n'avons pas lancé les acquisitions de l'IMU et du Qualisys en même temps, nous devions réussir à synchroniser les valeurs avant de pouvoir les comparer. Pour cela, nous avons cherché, pour chaque manipulation, un moment significatif qui nous servirait de point de référence. Nous avons donc décidé de travailler avec le moment du début du mouvement.

Pour le Qualisys, nous avons repéré ce moment sur le retour vidéo de l'expérience dans le logiciel QTM, à partir duquel il est possible d'obtenir la date (en secondes) du début du mouvement. Nous avons ensuite pu retrouver ce moment dans les données du Qualisys grâce au *timestamp* et déterminer la ligne de données à partir de laquelle il fallait étudier le mouvement.

Pour l'IMU, nous avons tracé les courbes, présentées ci-dessous, relatives à la rotation effectuée lors de chaque manipulation (roll, pitch ou yaw) grâce aux données. Puis, nous avons déterminé le moment de début du mouvement depuis ce graphique et nous l'avons retrouvé dans les données avec le *timestamp*.

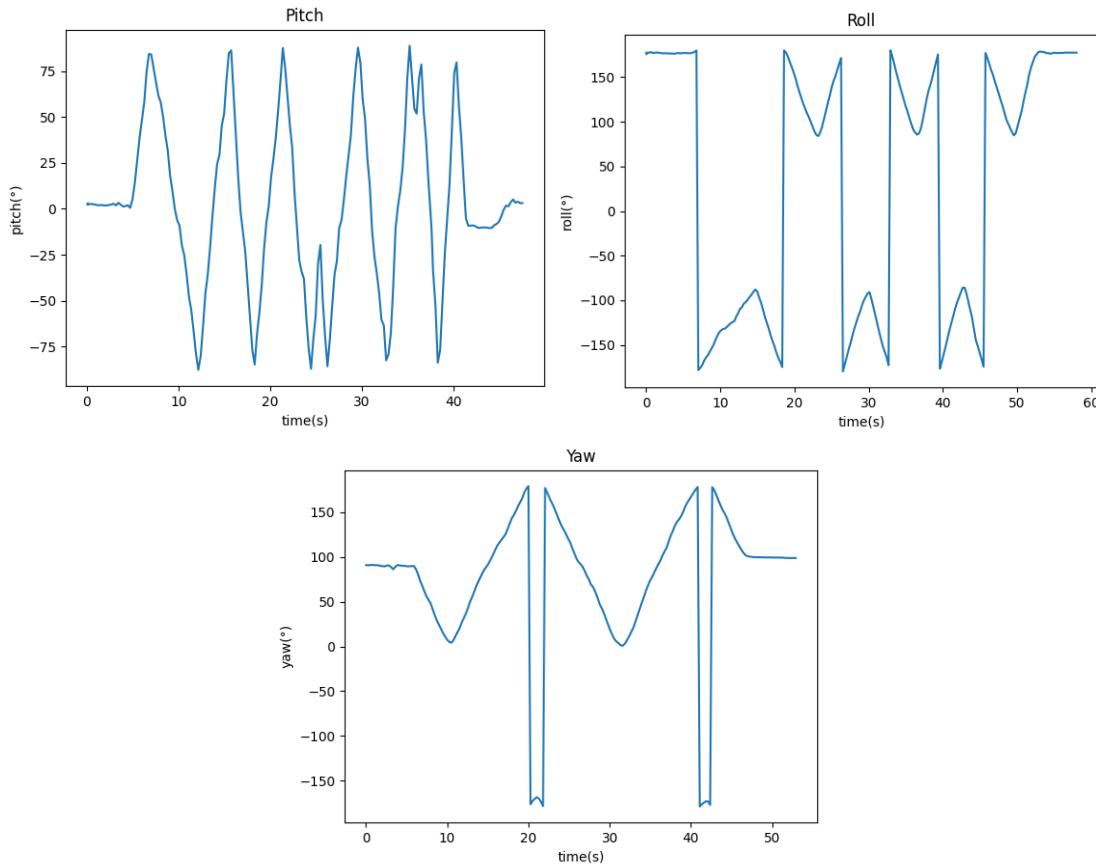


FIGURE 16 - Graphiques des 3 manipulations effectuées pour chaque axe de rotation
 (Roll : Perpendiculaire bâton 1 / Pitch : Axe bâton / Yaw : Plan sol)

En étudiant ces courbes et les vidéos Qualisys associées, nous avons pu confirmer que nous avons bien réussi à synchroniser les données. En effet, nous avons relevé les dates de fin du mouvement pour chaque manipulation et en comparant les temps totaux de manipulation mesurés par l'IMU et par le Qualisys, nous avons obtenu les écarts suivants figure 17.

	Total IMU (s)	Total Qualisys (s)	Ecart (s)
Roll 1	35.59	35.34	0.25
Roll 2	42.53	43.02	0.49
Pitch	35.60	35.30	0.30
Yaw	40.82	40.34	0.48

FIGURE 17 – Temps de manipulation totaux mesurés et écarts

On a donc au maximum un décalage de 0,5 secondes entre les valeurs de l'IMU et du Qualisys, ce qui est acceptable car nos mouvements étaient plutôt lents.

5.1.2 Fréquences de mesure différentes

Nous avons été confronté à un deuxième problème au moment de l'analyse des données relativ au fait que le système Qualisys et notre IMU ont des fréquences d'échantillonnage différentes.

En effet, le Qualisys est réglé par défaut sur 100Hz et l'IMU sur 4Hz. Ceci signifie que nous avons obtenu beaucoup plus de valeurs pour le Qualisys que pour l'IMU. Pour remédier à cela, comme nous n'avons pas pu refaire l'expérience, nous avons choisi d'extraire et d'analyser seulement une ligne de données du Qualisys sur 25 après avoir synchronisé les valeurs.

Pour de futures expériences, il est tout de même possible d'essayer de corriger ce problème avant les tests, ce que nous n'avons pas pu faire lors de notre projet.

En effet, dans la documentation du Phidget, nous avons trouvé une fonction permettant de choisir la fréquence d'échantillonnage du système. Nous avons donc vérifié la faisabilité de ce changement en essayant de nous aligner sur 100Hz avec le Qualisys. Cependant, lors des essais, nous nous sommes rendus compte que le Phidget acquiert une valeur toutes les 12ms au lieu de toutes les 10 ms, ce qui revient à une fréquence d'échantillonage de 83Hz au lieu de 100Hz. En cherchant dans la documentation, nous avons déterminé que la cause de notre problème pouvait être le *ChangeTrigger* associé à chaque channel de notre Phidget. Nous avons alors défini tous les *ChangeTrigger* à zéro mais cela n'a pas résolu le problème, et même si nous choisissons une fréquence d'échantillonage supérieure à 100Hz, le Phidget n'affiche une valeur au minimum que toutes les 12 ms. Pour obtenir la même fréquence, il faudrait donc essayer de baisser celle du Qualisys. Sinon, choisir une valeur de fréquence remarquable et le plus élevé possible, comme 75Hz, pour l'IMU et récupérer 3 valeurs sur 4 du Qualisys, ce qui reste mieux qu'une valeur sur 25.

5.1.3 Repères différents

Pour pouvoir valider nos manipulations, il faut que l'orientation du bâton donnée par l'IMU soit la même que celle déterminée par le système Qualisys. Or, les deux systèmes ont des repères monde différents, c'est-à-dire qu'ils calculent les angles de manière différentes. On considère donc le repère monde de l'IMU w_i et celui du Qualisys w_q . Ainsi, ce que l'on obtient grâce aux mesures deux systèmes sont :

- pour l'IMU : ${}^i R_{w_i}$ les matrices de rotation des valeurs mesurées par l'IMU dans son repère monde interne
- pour le Qualisys : ${}^q R_{w_q}$ les matrices de rotation des valeurs mesurées par le Qualisys dans son propre repère monde

Pour pouvoir comparer ces matrices, on souhaite donc, dans un premier temps, définir le repère de l'IMU dans le repère monde du Qualisys. On ne prend pas en compte ici les translations car seule l'orientation du système nous intéresse. On souhaite donc obtenir ${}^i R_{w_q}$, c'est-à-dire la matrice de rotation des valeurs mesurées par l'IMU dans le repère monde du Qualisys. Pour cela on peut écrire :

$${}^i R_{w_q} = {}^i R_{w_i} \cdot {}^{w_i} R_{w_q} \quad (4)$$

Le problème ici est que nous n'avons pas accès à la matrice ${}^{w_i} R_{w_q}$ car nous ne connaissons pas le repère monde du Qualisys. Il est donc impossible de déterminer la position du repère monde de l'IMU dans celui du Qualisys. Ainsi, cette méthode n'est pas utilisable.

Nous avons donc décidé de travailler par rapport à des valeurs initiales de position de l'IMU par rapport auxquelles nous étudierons le mouvement. Pour cela, nous choisissons une position

particulière (juste avant le début du mouvement) et on relève pour cet instant les valeurs initiales du positionnement de l'IMU dans son repère monde w_i et dans le système Qualisys w_q . Nous souhaitons obtenir au final que les rotations de l'IMU au cours du temps soit les mêmes dans son propre repère monde que dans celui du Qualisys. On étudie donc les rotations de l'IMU dans les deux repères par rapport à leurs positions initiales fixées plus tôt. On peut alors écrire que notre objectif est d'obtenir :

$${}^i R_{i_0} = {}^q R_{q_0} \quad (5)$$

Où,

- ${}^i R_{i_0}$ relative à la position courante de l'IMU par rapport à sa position initiale
- ${}^q R_{q_0}$ relative à la position courante de l'IMU dans le système Qualisys par rapport à sa position initiale

Si on utilise la relation de Chasles en sachant que l'IMU et le Qualisys donnent chacun des valeurs d'angles en fonction d'un repère monde qui leur est propre (respectivement w_i et w_q), on peut écrire :

$${}^i R_{w_i} \cdot {}^{w_i} R_{i_0} = {}^q R_{w_q} \cdot {}^{w_q} R_{q_0} \quad (6)$$

Où,

- ${}^i R_{w_i}$ est obtenue grâce aux valeurs mesurées par l'IMU
- ${}^q R_{w_q}$ est obtenue grâce aux valeurs mesurées par le Qualisys
- ${}^{w_i} R_{i_0}$ est relative à la position initiale de l'IMU dans son propre repère monde
- ${}^{w_q} R_{q_0}$ est relative à la position initiale de l'IMU dans le repère monde du Qualisys

5.2 Calcul des matrices de rotation

Nous avons décidé d'utiliser les quaternions pour calculer les matrices de rotation de l'IMU pour minimiser les erreurs de calculs, en particulier la conversion Euler-Quaternions et car les quaternions sont uniques à leur repère contrairement aux angles d'Euler.

Nous avons calculé les matrices de rotation ${}^i R_{w_i}$ à partir des valeurs de quaternions extraites de l'IMU par notre code Python. Les matrices de rotation ${}^q R_{w_q}$, quant à elles, sont obtenues en prenant la transposée des matrices ${}^{w_q} R_q$ résultant directement des données Qualisys.

Pour calculer une matrice de rotation, on utilise la formule suivante. Soit un quaternion $Q = a + ib + jc + kd$,

$$R = \begin{pmatrix} a^2 + b^2 - c^2 - d^2 & 2bc - 2ad & 2ac + 2bd \\ 2ad + 2bc & a^2 - b^2 + c^2 - d^2 & 2cd - 2ab \\ 2bd - 2ac & 2ab + 2cd & a^2 - b^2 - c^2 + d^2 \end{pmatrix} \quad (7)$$

Une fois les matrices ${}^i R_{w_i}$ calculées, nous avons aussi déterminé, pour chaque manipulation, la matrice de rotation initiale ${}^{w_i} R_{i_0}$ ($= {}^t({}^{i_0} R_{w_i})$) à partir des premières données de l'IMU (correspondant au début du mouvement comme convenu plus tôt). De la même manière, pour le Qualisys, nous avons fixé la première matrice de rotation des données comme ${}^{w_q} R_{q_0}$.

Une attention particulière doit être accordée aux matrices de rotation et leurs transposées car la Qualisys calcule des matrices de la forme ${}^{w_q} R_q$, alors que les matrices calculées avec la formule (7) pour les données de l'IMU, sont de la forme ${}^i R_{w_i}$.

Enfin, nous avons multiplié les matrices, respectivement relatives à l'IMU et au Qualisys, afin d'obtenir les termes de gauche et de droite de l'équation (6) et de vérifier s'il s'agit bien d'une égalité.

Le code MATLAB exécutant ces calculs est donné en annexe 7.7.

5.3 Comparaison des matrices

Nous avons posé un écart maximal $\epsilon = 0.1$ entre les valeurs des matrices en dessous duquel nous considérons les valeurs comme égales. En effet, un ϵ à 0.1 équivaut à un pourcentage d'erreur de 5% car les valeurs des matrices sont comprises entre -1 et 1. Le but de cette comparaison est d'avoir le maximum de valeurs possibles dans un intervalle de confiance à 95%.

Le code MATLAB effectuant cette comparaison est donné en annexe 7.8.

Les résultats obtenus pour les quatre manipulations sont présentés figure 18 suivante.

	Valeurs correctes	Valeurs incorrectes
Roll 1	39.2%	60.8%
Roll 2	29.8%	70.2%
Pitch	32.9%	67.1%
Yaw	76.5%	23.5%

FIGURE 18 – Pourcentages de réussite et d'erreur

On remarque alors que seule la quatrième manipulation semble donner des résultats corrects à plus de 75%. Les autres ont une grande majorité de valeurs incorrectes, ce qui veut que l'équation (6) n'est pas respectée.

Nous avons remarqué que le pitch de l'IMU est très perturbé lorsque l'on s'approche de ces valeurs limites (au delà de -80° et 80°), ceci pourrait expliquer pourquoi ses valeurs sont incorrectes. Nous avions également observé ce comportement lors de nos tests initiaux partie 3.2.2

5.4 Visualisation des rotations

Nous avons alors tenté de visualiser les rotations effectuées dans chaque repère (IMU et Qualisys) pour obtenir plus de raisons quant à nos précédents mauvais résultats. Pour cela, nous

avons créé des vidéos GIF, comme présenté figure 19, qui montrent à gauche les rotations de l'IMU perçues dans son repère monde interne, et à droite les rotations de l'IMU perçues par les Qualisys. Si l'équation 6 est respectée, les cubes devraient tourner en même temps, sur le même axe et dans le même sens.

Le code MATLAB créant les vidéos GIF est donné en annexe 7.9.

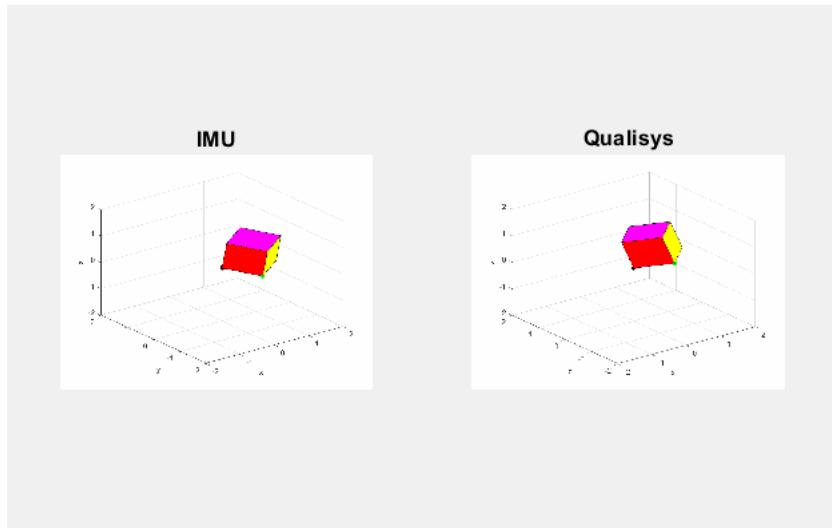


FIGURE 19 – Interface de visualisation des rotations IMU et Qualisys (exemple du Yaw)

Dans ces vidéos, on remarque dans un premier temps que dans les deux repères les rotations sont bien réalisées de manière synchrone, seulement elles ne s'effectuent pas sur les mêmes axes de rotation, excepté le yaw pour lequel l'axe z est commun aux deux repères. Ainsi, il existe une **différence de convention** entre les deux repères. Cette différence est la cause principale des erreurs calculées dans la partie précédente. En effet, comme pour le yaw, si les autres rotations s'effectuaient sur le même axe, l'égalité (6) serait presque atteinte.

Le deuxième point que l'on remarque est qu'au bout d'un certain temps, le plan de l'IMU dans lequel s'effectue la rotation semble changer contrairement à celui du Qualisys. Ceci peut être expliqué par le fait que l'on a **désactivé les magnétomètres**, donc l'IMU se fie à ses accéléromètres et gyroscopes pour déterminer son cap, c'est pourquoi il a du mal à le maintenir sur le long terme et qu'il se perd au bout d'un moment, surtout lorsque l'on bouge rapidement le système.

Ces vidéos ont tout de même en grande partie permis de valider les données recueillies par l'IMU. En effet, bien que celle-ci se dérègle au cours du temps, ce problème sera résolu en mer grâce à la lenteur des mouvements et aux magnétomètres. De plus, le problème de convention des repères peut lui aussi être résolu. Dans l'ensemble, l'IMU parvient bien à acquérir les rotations du système en temps réel.

6 Conclusion

Au cours de ce projet nous avons pu prendre en main une centrale inertuelle Phidget, découvrir et appliquer la librairie *Phidget2022* qui lui est associée. De plus, nous avons pu réaliser plusieurs manipulations avec le système Qualisys et travailler avec le logiciel Qualisys Track Manager.

Ce projet aura finalement permis de montrer que la centrale inertuelle Phidget sans ses magnétomètres parvient en grande partie à déterminer l'orientation et les rotations d'un objet en temps réel, du moment que ce dernier ne se déplace pas trop vite. En effet, sinon il a tendance à perdre son cap. Ce problème sera résolu en milieu marin car les mouvements dans l'eau seront beaucoup plus lents que dans l'air et les magnétomètres seront enclenchés.

Par ailleurs, pour avoir des résultats réellement concluants du point de vue des repères de l'IMU et du Qualisys, il faudrait réussir à comprendre quelle est la convention du Qualisys, afin d'orienter les deux repères de la même manière et ne pas avoir des rotations sur des axes différents. Ainsi, l'égalité de l'équation 6 serait beaucoup plus respectée et nos calculs de pourcentages donneraient pour toutes les manipulations des résultats correctes comme pour le yaw. De plus, la similitude des données IMU et Qualisys seraient beaucoup plus évidentes sur les vidéos GIF.

Enfin, nous n'avons pu réalisé qu'une seule fois les manipulations car nous n'avons pas eu l'occasion d'utiliser le système Qualisys à nouveau. Ainsi, certains autres aspects de manipulation moins essentiels pourraient aussi modifiés pour obtenir des données plus simples à analyser et plus nombreuses :

- Fixer la fréquence de l'IMU au plus proche de celle du Qualisys à 100 Hz, pour avoir des valeurs plus proches dans le temps et pouvoir analyser toutes la données sans avoir à en couper
- Lancer le code de l'IMU et QTM en même temps sur deux ordinateurs différents, pour la synchronisation
- Poser le bâton au sol ou sur une table au début des manipulations, pour fixer un repère de référence fiable et immobile

7 Annexes

7.1 Code Python : Exécution des Phidgets et écriture des données en temps réel : *Phidget.py*

```

1  from Phidget22.Phidget import *
2  from Phidget22.Devices.Spatial import *
3  from Phidget22.Devices.TemperatureSensor import *
4  from Phidget22.Devices.Magnetometer import *
5  import time
6  import csv

7
8 ##### fonction creant le fichier texte pour sauvegarder les donnees en #####
9 ##### temps reel du phidget 1 #####
10 ##### temps reel du phidget 2 #####
11 #####
12 #####
13 def CSVfilecreation() :
14     fields = [ 'timestamp' , 'pitch' , 'roll' , 'yaw' , 'QuaternionX' ,
15                'QuaternionY' , 'QuaternionZ' , 'QuaternionW' ]
16     with open( 'CSVfiletest' , 'w' ) as csvdoc:
17         write = csv.writer(csvdoc)
18         write.writerow(fields)
19         csvdoc.close

20
21 #####
22 ##### fonction creant le fichier texte pour sauvegarder les donnees en #####
23 ##### temps reel du phidget 2 #####
24 #####
25 #####
26 #####
27 def CSVfilecreation1() :
28     fields = [ 'timestamp' , 'pitch' , 'roll' , 'yaw' , 'QuaternionX' ,
29                'QuaternionY' , 'QuaternionZ' , 'QuaternionW' ]
30     with open( 'CSVfiletest1' , 'w' ) as csvdoc:
31         write = csv.writer(csvdoc)
32         write.writerow(fields)
33         csvdoc.close

34 #####
35 ##### fonction permettant d'ecrire dans le fichier texte du phidget 1 #####
36 ##### fonction permettant d'ecrire dans le fichier texte du phidget 2 #####
37 #####
38 #####
39 def CSVWriter(timestamp , pitch , roll , yaw , qx , qy , qz , qw) :
40     rows = [ timestamp , pitch , roll , yaw , qx , qy , qz , qw]
41     with open( 'CSVfiletest' , 'a' ) as csvdoc:
42         write = csv.writer(csvdoc)
43         write.writerow(rows)
44         csvdoc.close

45 #####
46 ##### fonction permettant d'ecrire dans le fichier texte du phidget 2 #####
47 #####
48 #####
49 #####

```

```

50 def CSVWriter1(timestamp, pitch, roll, yaw, qx, qy, qz, qw) :
51     rows = [timestamp, pitch, roll, yaw, qx, qy, qz, qw]
52     with open('CSVfiletest1', 'a') as csvdoc:
53         write = csv.writer(csvdoc)
54         write.writerow(rows)
55     csvdoc.close
56
57 #####
58 ##### fonction permettant d'afficher et d'écrire les données du phidget 1 #####
59 ##### #####
60 ##### #####
61
62 def onAlgorithmData(self, quaternion, timestamp):
63     #print("Timestamp: " + str(timestamp))
64
65     eulerAngles = self.getEulerAngles()
66
67     #print("EulerAngles: ")
68     print("\tpitch: " + str(eulerAngles.pitch))
69     print("\troll: " + str(eulerAngles.roll))
70     print("\thead: " + str(eulerAngles.heading))
71
72     quaternion = self.getQuaternion()
73     #print("Quaternion: ")
74     #print("\tx: " + str(quaternion.x))
75     #print("\ty: " + str(quaternion.y))
76     #print("\tz: " + str(quaternion.z))
77     #print("\tw: " + str(quaternion.w))
78     #print("-----")
79
80
81
82     CSVWriter(timestamp, eulerAngles.pitch, eulerAngles.roll,
83               eulerAngles.heading, quaternion.x, quaternion.y,
84               quaternion.z, quaternion.w)
85
86 ##### fonction permettant d'afficher et d'écrire les données du phidget 2 #####
87 ##### #####
88 ##### #####
89
90 def onAlgorithmData1(self, quaternion, timestamp):
91     #print("Timestamp: " + str(timestamp))
92
93     eulerAngles = self.getEulerAngles()
94
95     #print("EulerAngles: ")
96     #print("\tpitch: " + str(eulerAngles.pitch))
97     #print("\troll1: " + str(eulerAngles.roll))
98     #print("\thead: " + str(eulerAngles.heading))
99
100    quaternion = self.getQuaternion()
101    #print("Quaternion: ")
102    #print("\tx: " + str(quaternion.x))
103    #print("\ty: " + str(quaternion.y))
104    #print("\tz: " + str(quaternion.z))
105    #print("\tw: " + str(quaternion.w))

```

```

106     #print ("-----")
107
108
109
110     CSVWriter1(timestamp , eulerAngles . pitch , eulerAngles . roll ,
111                 eulerAngles . heading , quaternion .x ,
112                 quaternion .y , quaternion .z , quaternion .w)
113
114 ##### Fonction Main #####
115 ##### Fonction Main #####
116 ##### Fonction Main #####
117
118 def main():
119
120     ### Initialisation et association au phidget 1 par son numero de serie ###
121
122         spatial0 = Spatial()
123         temperature0=TemperatureSensor()
124         magnetometer0=Magnetometer()
125         spatial0.setDeviceSerialNumber(373654)
126         temperature0.setDeviceSerialNumber(373654)
127         magnetometer0.setDeviceSerialNumber(373654)
128
129     ### Initialisation et association au phidget 2 par son numero de serie ###
130
131         spatial1 = Spatial()
132         temperature1=TemperatureSensor()
133         magnetometer1=Magnetometer()
134         spatial1.setDeviceSerialNumber(373141)
135         temperature1.setDeviceSerialNumber(373141)
136         magnetometer1.setDeviceSerialNumber(373141)
137
138     ### Affichage et écriture des donnees des phidgets
139
140         spatial0.setOnAlgorithmDataHandler(onAlgorithmData)
141         spatial1.setOnAlgorithmDataHandler(onAlgorithmData1)
142
143     ### ouverture des channels ###
144
145         temperature0.open()
146         magnetometer0.open()
147         temperature1.open()
148         magnetometer1.open()
149         spatial0.openWaitForAttachment(1000)
150         spatial1.openWaitForAttachment(1000)
151
152     ### Fixe le Change Trigger pour les magnetos et thermometres ###
153     ### des 2 phidgets ###
154
155         temperature0.setTemperatureChangeTrigger(0)
156         magnetometer0.setMagneticFieldChangeTrigger(0)
157         temperature1.setTemperatureChangeTrigger(0)
158         magnetometer1.setMagneticFieldChangeTrigger(0)
159
160     ### Fixe intervalle d'echantillonage a 100 Hz ###
161

```

```

162     spatial0.setDataRate(100)
163     spatial1.setDataRate(100)
164     #spatial0.setDataInterval(10) # en ms
165
166
167
168     ### Desactive l'utilisation du magnetometre pour      ###
169     ### simuler un environment ou le magneto est interdit  ###
170
171     spatial0.setAlgorithmMagnetometerGain(0.0000005)
172     spatial1.setAlgorithmMagnetometerGain(0.0000005)
173
174
175     ### Ajout d'une fonction appui sur une touche ###
176     ### pour mettre fin au programme   ###
177
178     try:
179         input("Press Enter to Stop\n")
180     except (Exception, KeyboardInterrupt):
181         pass
182
183     ### fermeture des channels ###
184
185     temperature0.close()
186     magnetometer0.close()
187     temperature1.close()
188     magnetometer1.close()
189     spatial0.close()
190     spatial1.close()
191
192 ######
193 ######
194 ######
195 ######
196
197 CSVfilecreation()
198 CSVfilecreation1()
199 print("Lancement effectue")
200 main()

```

7.2 Code Python : Extraction des données Euler depuis les fichiers IMU : *EulerFile_IMU.py*

```

1 import csv
2 import numpy as np
3
4 ######
5 ## fonction creant le fichier texte pour sauvegarder les donnees ##
6 ######
7
8 def CSVfilecreation() :
9     with open('EulerFile_IMU.txt', 'w') as csvdoc:
10        write = csv.writer(csvdoc)

```

```

11         csvdoc.close
12 #####
13 ##### fonction qui compte les lignes d'un fichier texte/tsv #####
14 ##### #####
15 #####
16
17 def Comptagedeligne(fichier) :
18     with open(fichier, 'r') as f:
19         obj=csv.reader(f)
20         compteur=0
21         for ligne in obj :
22             compteur=compteur+1
23     return(compteur)
24
25 #####
26 ##### fonction permettant d'ecrire dans le fichier texte #####
27 ##### #####
28
29 def CSVWriter(roll,pitch,yaw) :
30     rows = [roll,pitch,yaw]
31     with open('EulerFile_IMU.txt','a') as csvdoc:
32         write = csv.writer(csvdoc)
33         write.writerow(rows)
34     csvdoc.close
35
36 #####
37 ##### Main #####
38 #####
39
40 CSVfilecreation()
41
42 ##### demande a l'utilisateur le nom du fichier texte contenant #####
43 ##### les donnees du phidget et la ligne correspondant au t0 #####
44
45 print('Entrer le nom du fichier contenant les donnees Euler IMU :')
NomFichier = input()
46 print('Entrer le numero de la ligne correspondant a t0 :')
NumLigneDebut = input()
47 NumLigneDebut = int(NumLigneDebut)
48 NumLigneDebut = NumLigneDebut-1
49
50
51 print('Entrer le numero de la ligne correspondant a tf :')
NumLigneFin = input()
52 NumLigneFin = int(NumLigneFin)
53 NumLigneFin = NumLigneFin
54
55
56 ##### ouverture du fichier en mode lecture pour creation differents #####
57 ##### vecteurs et remplissage #####
58
59
60 with open(NomFichier, 'r') as f:
61     obj=csv.reader(f)
62     compteur=Comptagedeligne(NomFichier)
63
64     roll=np.zeros(compteur-NumLigneDebut-(compteur-NumLigneFin)+1)
65     pitch=np.zeros(compteur-NumLigneDebut-(compteur-NumLigneFin)+1)
66     yaw=np.zeros(compteur-NumLigneDebut-(compteur-NumLigneFin)+1)

```

```

67     i=0
68     for ligne in obj:
69         if i >= NumLigneDebut and i <= NumLigneFin:
70             roll[i-NumLigneDebut]=float(ligne[1])
71             pitch[i-NumLigneDebut]=float(ligne[2])
72             yaw[i-NumLigneDebut]=float(ligne[3])
73             i=i+1
74
75     ### Ecriture dans un fichier des donnees phidgets recuperees ###
76
77     for i in range (len(roll)) :
78         CSVWriter(roll[i],pitch[i],yaw[i])

```

7.3 Code Python : Extraction des données Quaternions depuis les fichiers IMU : *QuaternionsFile_IMU.py*

```

1 import csv
2 import numpy as np
3
4 ##### fonction creant le fichier texte pour sauvegarder les donnees #####
5 ##### fonction permettant de compter les lignes d'un fichier texte ou tsv #####
6 ##### fonction permettant d'ecrire dans le fichier texte #####
7
8 def CSVfilecreation() :
9     with open('QuaternionFile_IMU.txt','w') as csvdoc:
10        write = csv.writer(csvdoc)
11        csvdoc.close
12
13 ##### fonction permettant de compter les lignes d'un fichier texte ou tsv #####
14 ##### fonction permettant de compter les lignes d'un fichier texte ou tsv #####
15 ##### fonction permettant de compter les lignes d'un fichier texte ou tsv #####
16
17 def Comptagedeligne(fichier) :
18     with open(fichier,'r') as f:
19         obj=csv.reader(f)
20         compteur=0
21         for ligne in obj :
22             compteur=compteur+1
23     return (compteur)
24
25 ##### fonction permettant d'ecrire dans le fichier texte #####
26 ##### fonction permettant d'ecrire dans le fichier texte #####
27 ##### fonction permettant d'ecrire dans le fichier texte #####
28
29 def CSVWriter(qx,qy,qz,qw) :
30     rows = [qx,qy,qz,qw]
31     with open('QuaternionFile_IMU.txt','a') as csvdoc:
32         write = csv.writer(csvdoc)
33         write.writerow(rows)
34         csvdoc.close
35
36 ##### Main #####
37 ##### Main #####

```

```

38 ##########
39
40 CSVfilecreation()
41
42 ##### demande a l'utilisateur le nom du fichier texte contenant #####
43 ##### les donnees du phidget et la ligne correspondant au t0 #####
44
45 print('Entrer le nom du fichier contenant les donnees quaternions IMU :')
46 NomFichier = input()
47 print('Entrer le numero de la ligne correspondant a t0 :')
48
49 NumLigneDebut = input()
50 NumLigneDebut = int(NumLigneDebut)
51 NumLigneDebut = NumLigneDebut - 1
52
53 ##### ouverture du fichier en mode lecture pour creation #####
54 ##### differents vecteurs et remplissage #####
55
56 with open(NomFichier, 'r') as f:
57     obj=csv.reader(f)
58     compteur=Comptagedeligne(NomFichier)
59
60     QuaternionX=np.zeros(compteur-NumLigneDebut)
61     QuaternionY=np.zeros(compteur-NumLigneDebut)
62     QuaternionZ=np.zeros(compteur-NumLigneDebut)
63     QuaternionW=np.zeros(compteur-NumLigneDebut)
64     i=0
65     for ligne in obj:
66         if i >= NumLigneDebut:
67             QuaternionX[i-NumLigneDebut]=float(ligne[4])
68             QuaternionY[i-NumLigneDebut]=float(ligne[5])
69             QuaternionZ[i-NumLigneDebut]=float(ligne[6])
70             QuaternionW[i-NumLigneDebut]=float(ligne[7])
71         i=i+1
72
73 ##### Ecriture dans un fichier des donnees phidgets recuperees #####
74
75 for i in range(len(QuaternionX)) :
76     CSVWriter(QuaternionX[i],QuaternionY[i],QuaternionZ[i],QuaternionW[i])

```

7.4 Code Python : Tracé des courbes des angles d'Euler de l'IMU : *plot.py*

```

1 import csv
2 import numpy as np
3 import matplotlib as mp
4 import matplotlib.pyplot as plt
5
6 ##########
7 ##### fonction permettant de compter les lignes d'un fichier texte ou tsv #####
8 ##########
9
10 def Comptagedeligne(fichier) :

```

```

11     with open(fichier, 'r') as f:
12         obj=csv.reader(f)
13         compteur=0
14         for ligne in obj :
15             compteur=compteur+1
16         return(compteur)
17
18 ##### Fonction Main #####
19 ##### Fonction Main #####
20 ##### Fonction Main #####
21
22 def main() :
23
24     """ demande a l'utilisateur le nom du fichier texte contenant """
25     """ les donnees pour les courbes """
26
27     print('Entrer le nom du fichier :')
28     NomFichier = input()
29
30     """ ouverture du fichier en mode lecture pour creation """
31     """ differents vecteurs et leur remplissage """
32
33     with open(NomFichier, 'r') as f:
34         obj=csv.reader(f)
35         compteur=Comptagedeligne(NomFichier)
36         timestamp=np.zeros(compteur-1)
37         pitch=np.zeros(compteur-1)
38         roll=np.zeros(compteur-1)
39         yaw=np.zeros(compteur-1)
40         QuaternionX=np.zeros(compteur-1)
41         QuaternionY=np.zeros(compteur-1)
42         QuaternionZ=np.zeros(compteur-1)
43         QuaternionW=np.zeros(compteur-1)
44         i=0
45         for ligne in obj:
46             if i !=0:
47
48                 #tps en secondes
49                 timestamp[i-1]=float(ligne[0])/1000
50
51                 # recuperation des donnees
52
53                 pitch[i-1]=float(ligne[1])
54                 roll[i-1]=float(ligne[2])
55                 yaw[i-1]=float(ligne[3])
56                 #QuaternionX[i-1]=float(ligne[4])
57                 #QuaternionY[i-1]=float(ligne[5])
58                 #QuaternionZ[i-1]=float(ligne[6])
59                 #QuaternionW[i-1]=float(ligne[7])
60
61                 i=i+1
62
63             # trace des differentes courbes
64
65             plt.plot(timestamp,pitch)
66             plt.xlabel("time(s)")
67             plt.ylabel("pitch(deg)")

```

```

67     plt.title("Pitch")
68     plt.savefig("Pitch.png")
69     plt.show()
70
71     plt.plot(timestamp, roll)
72     plt.xlabel("time(s)")
73     plt.ylabel("roll(deg)")
74     plt.title("Roll")
75     plt.savefig("Roll.png")
76     plt.show()
77
78     plt.plot(timestamp, yaw)
79     plt.xlabel("time(s)")
80     plt.ylabel("yaw(deg)")
81     plt.title("Yaw")
82     plt.savefig("Yaw.png")
83     plt.show()
84
85 ######
86 ######
87 ######
88 ######
89 main()

```

7.5 Code Python : Calcul des données Quaternions Qualisys depuis les données Euler Qualisys : *QuaternionsFile_Qualisys.py*

```

1 import csv
2 import numpy as np
3 import pandas as pd
4
5 ######
6 ### fonction creant le fichier texte pour sauvegarder les donnees quaternions ###
7 ######
8
9 def CSVfilecreation_Quaternion() :
10     with open('QuaternionFile_Qualisys.txt', 'w') as csvdoc:
11         write = csv.writer(csvdoc)
12         csvdoc.close()
13
14 ######
15 ### fonction creant le fichier texte intermediaire permettant ##
16 ### de faire le tri des donnees ## #
17 ######
18
19 def CSVfilecreation_EulerTri() :
20     with open('QuaternionFile_EulerTri.txt', 'w') as csvdoc:
21         write = csv.writer(csvdoc)
22         csvdoc.close()
23
24 ######
25 ### fonction permettant de compter les lignes d'un fichier texte/tsv ##
26 ######

```

```

27
28 def Comptagedeligne(fichier) :
29     with open(fichier , 'r') as f:
30         obj=csv . reader(f)
31         compteur=0
32         for ligne in obj :
33             compteur=compteur+1
34     return(compteur)
35
36 ##### fonction permettant d'ecrire dans le fichier contenant les quaternions#####
37 #### fonction permettant d'ecrire dans le fichier contenant les donnees Euler triees #####
38 #####
39
40 def CSVWriter_QuaternionFile(qx ,qy ,qz ,qw) :
41     rows = [qx ,qy ,qz ,qw]
42     with open( 'QuaternionFile_Qualisys .txt' , 'a' ) as csvdoc:
43         write = csv . writer(csvdoc)
44         write . writerow(rows)
45     csvdoc . close
46
47 ##### fonction permettant d'ecrire dans le fichier contenant #####
48 #### fonction permettant d'ecrire dans le fichier contenant #####
49 #### les donnees Euler triees #####
50 #####
51
52 def CSVWriter_EulerTri(roll ,pitch ,yaw) :
53     rows = [roll ,pitch ,yaw]
54     with open( 'QuaternionFile_EulerTri .txt' , 'a' ) as csvdoc:
55         write = csv . writer(csvdoc)
56         write . writerow(rows)
57     csvdoc . close
58
59 ##### fonction permettant de trier les donnees Euler : Passage de 100 Hz a 4Hz #####
60 #####
61 #####
62
63
64 def TSVFile2EulerTriFile(NomFichierTSV , DataFrameFile ,NumLigneDebut , NumLigneFin) :
65     compteur=Comptagedeligne(NomFichierTSV)
66
67     reste = (compteur-NumLigneDebut -(compteur-NumLigneFin)+1)%25
68     taille= (compteur-NumLigneDebut -(compteur-NumLigneFin)+1-reste)/25
69     taille=int(taille)+1
70
71     roll=np . zeros(taille )
72     pitch=np . zeros(taille )
73     yaw=np . zeros(taille )
74     i=0
75     for j in range(compteur -1):
76         if j >= NumLigneDebut and j<=NumLigneFin:
77             if j%25 == 0 :
78
79                 roll [i]= float(DataFrameFile . iat[j ,5])
80                 pitch [i]= float(DataFrameFile . iat[j ,6])
81                 yaw [i]= float(DataFrameFile . iat[j ,7])
82                 i=i+1

```

```

83
84     for i in range (len(roll)) :
85         CSVWriter_EulerTri(roll[i],pitch[i],yaw[i])
86
87 ##### fonction permettant de passer des angles d'Euler aux quaternions #####
88 #### Ecriture dans le fichier contenant les donnees Quaternions Qualisys #####
89 #####
90 #####
91 #####
92 def Euler2Quaternion(roll , pitch , yaw):
93
94     Qx = np.sin(roll/2)*np.cos(pitch/2)*np.cos(yaw/2) - np.cos(roll/2)*
95         np.sin(pitch/2)*np.sin(yaw/2)
96     Qy = np.cos(roll/2)*np.sin(pitch/2)*np.cos(yaw/2) + np.sin(roll/2)*
97         np.cos(pitch/2)*np.sin(yaw/2)
98     Qz = np.cos(roll/2)*np.cos(pitch/2)*np.sin(yaw/2) - np.sin(roll/2)*
99         np.sin(pitch/2)*np.cos(yaw/2)
100    Qw = np.cos(roll/2)*np.cos(pitch/2)*np.cos(yaw/2) + np.sin(roll/2)*
101        np.sin(pitch/2)*np.sin(yaw/2)
102
103
104     CSVWriter_QuaternionFile(Qx,Qy,Qz,Qw)
105
106 ##### fonction permettant de convertir des degres en radians #####
107 #####
108 #####
109 #####
110 def Deg2Rad(Deg_angle):
111     Rad_angle=Deg_angle*np.pi/180.00
112     return(Rad_angle)
113
114 #####
115 ##### Fonction Main #####
116 #####
117 #####
118 def main():
119
120     #### demande a l'utilisateur le nom du fichier texte contenant #####
121     #### les donnees du phidget et la ligne correspondant au t0 #####
122
123     print('Entrer le nom du fichier contenant les donnees Euler Qualisys :')
124     NomFichier = input()
125     Fichier=pd.read_csv(NomFichier, sep='\t')
126     print('Entrer le numero de la ligne correspondant a t0 :')
127     NumLigneDebut = input()
128     NumLigneDebut = int(NumLigneDebut)
129     NumLigneDebut = NumLigneDebut-1
130
131     print('Entrer le numero de la ligne correspondant a tf :')
132     NumLigneFin = input()
133     NumLigneFin = int(NumLigneFin)
134     NumLigneFin = NumLigneFin-1
135
136     #### Passe d'un echantillonage 100 Hz a un echantillonage 4Hz ####
137
138     TSVFile2EulerTriFile(NomFichier,Fichier, NumLigneDebut, NumLigneFin)

```

```

139
140     ### ouverture du fichier en mode lecture pour creation #####
141     ##### differents vecteurs et remplissage #####
142
143     with open( 'QuaternionFile_EulerTri.txt' , 'r' ) as f:
144         obj=csv.reader(f)
145         compteur1=Comptagedeligne( 'QuaternionFile_EulerTri.txt' )
146         print(compteur1)
147
148         pitch_1=np.zeros(compteur1)
149         roll_1=np.zeros(compteur1)
150         yaw_1=np.zeros(compteur1)
151
152         i=0
153         for ligne in obj:
154             roll_1[i]=float(ligne[0])
155             pitch_1[i]=float(ligne[1])
156             yaw_1[i]=float(ligne[2])
157             i=i+1
158
159     ### passage des angles d'euler aux quaternions et ecriture dans le fichier #####
160
161     for i in range (len(roll_1)) :
162         Rad_Roll=Deg2Rad(roll_1[i])
163         Rad_Pitch=Deg2Rad(pitch_1[i])
164         Rad_Yaw=Deg2Rad(yaw_1[i])
165
166         Euler2Quaternion(Rad_Roll, Rad_Pitch, Rad_Yaw)
167
168 #####
169 #####
170
171 CSVfilecreation_Quaternion()
172 CSVfilecreation_EulerTri()
173 main()

```

7.6 Code Python : Extraction des données des matrices de rotations depuis les fichiers Qualisys : *RotFile_Qualisys.py*

```

1 import csv
2 import numpy as np
3 import pandas as pd
4
5 #####
6 #### fonction creant le fichier texte pour sauvegarder les donnees quaternions #####
7 #####
8
9 def CSVfilecreation_Rot() :
10     with open('RotFile_Qualisys.txt', 'w') as csvdoc:
11         write = csv.writer(csvdoc)
12         csvdoc.close
13

```

```

14 #####
15 #####fonction permettant de compter les lignes d'un fichier texte ou tsv #####
16 #####fonction permettant de compter les lignes d'un fichier texte ou tsv #####
17 #####fonction permettant de compter les lignes d'un fichier texte ou tsv #####
18
19 def Comptagedeligne(fichier) :
20     with open(fichier , 'r') as f:
21         obj=csv . reader(f)
22         compteur=0
23         for ligne in obj :
24             compteur=compteur+1
25     return (compteur)
26
27 #####fonction permettant d'ecrire dans le fichier contenant les rotations #####
28 #####fonction permettant d'ecrire dans le fichier contenant les rotations #####
29 #####fonction permettant d'ecrire dans le fichier contenant les rotations #####
30
31 def CSVWriter_RotFile(rot0 ,rot1 ,rot2 ,rot3 ,rot4 ,rot5 ,rot6 ,rot7 ,rot8) :
32     rows = [rot0 ,rot1 ,rot2 ,rot3 ,rot4 ,rot5 ,rot6 ,rot7 ,rot8]
33     with open( 'RotFile_Qualisys.txt' , 'a' ) as csvdoc:
34         write = csv . writer(csvdoc)
35         write . writerow(rows)
36         csvdoc . close
37
38 #####Recuperation des donnees des matrices de rotations #####
39 #####Recuperation des donnees des matrices de rotations #####
40 #####Recuperation des donnees des matrices de rotations #####
41
42
43 def TSVFile2RotFile(NomFichierTSV , DataFrameFile ,NumLigneDebut , NumLigneFin) :
44
45     compteur=Comptagedeligne(NomFichierTSV)
46
47     reste = (compteur -NumLigneDebut -(compteur -NumLigneFin)+1)%25
48     taille= (compteur -NumLigneDebut -(compteur -NumLigneFin)+1- reste )/25
49     taille=int(taille)+1
50
51     rot0=np . zeros(taille )
52     rot1=np . zeros(taille )
53     rot2=np . zeros(taille )
54     rot3=np . zeros(taille )
55     rot4=np . zeros(taille )
56     rot5=np . zeros(taille )
57     rot6=np . zeros(taille )
58     rot7=np . zeros(taille )
59     rot8=np . zeros(taille )
60     i=0
61
62     for j in range(compteur -1):
63         if j >= NumLigneDebut and j<=NumLigneFin:
64
65             rot0 [i]=float(DataFrameFile . iat[j ,9])
66             rot1 [i]=float(DataFrameFile . iat[j ,10])
67             rot2 [i]=float(DataFrameFile . iat[j ,11])
68             rot3 [i]=float(DataFrameFile . iat[j ,12])
69             rot4 [i]=float(DataFrameFile . iat[j ,13])

```

```

70     rot5[i]=float(DataFrameFile.iat[j,14])
71     rot6[i]=float(DataFrameFile.iat[j,15])
72     rot7[i]=float(DataFrameFile.iat[j,16])
73     rot8[i]=float(DataFrameFile.iat[j,17])
74     i=i+1
75
76     for i in range(len(rot0)) :
77         CSVWriter_RotFile(rot0[i],rot1[i],rot2[i],rot3[i],
78                             rot4[i],rot5[i],rot6[i],rot7[i],rot8[i])
79
80 ##### Fonction Main #####
81 #####
82 #####
83
84 def main():
85
86     ## demande a l'utilisateur le nom du fichier texte contenant les ##
87     ## donnees du phidget et la ligne correspondant au t0 ##
88
89     print('Entrer le nom du fichier contenant les donnees Rot Qualisys :')
90     NomFichier = input()
91     Fichier=pd.read_csv(NomFichier, sep='\t')
92     print('Entrer le numero de la ligne correspondant a t0 :')
93     NumLigneDebut = input()
94     NumLigneDebut = int(NumLigneDebut)
95     NumLigneDebut = NumLigneDebut-1
96
97     print('Entrer le numero de la ligne correspondant a tf :')
98     NumLigneFin = input()
99     NumLigneFin = int(NumLigneFin)
100    NumLigneFin = NumLigneFin-1
101
102    ### Ecriture des rot dans le fichier ###
103
104    TSVFile2RotFile(NomFichier,Fichier, NumLigneDebut, NumLigneFin)
105
106
107 ##### CSVfilecreation_Rot #####
108 #####
109
110 CSVfilecreation_Rot()
111 main()

```

7.7 Code MATLAB : Matrice de rotation

```

1 clear
2 clc
3 close all
4
5
6 %% Chargement des donnees %%
7
8 %%--IMU--% Chargement des quaternions IMU
9 data_IMU=load("IMU_PlanSol_Baton_Lent_4.txt");

```

```

10 n=length(data_IMU(:,1));
11 %--Qualisys--% Chargement des matrices de rotation Qualisys
12 data_Q=load("RotFile_Qualisys_4.txt");
13 m=length(data_Q(:,1));
14
15
16 %% Matrices de rotation initiales %%
17
18 %%--IMU--%
19 quat0_IMU=data_IMU(1,:);
20 w_i0=quat0_IMU(4);
21 x_i0=quat0_IMU(1);
22 y_i0=quat0_IMU(2);
23 z_i0=quat0_IMU(3);
24
25 M0_IMU = [ w_i0*w_i0+x_i0*x_i0-y_i0*y_i0-z_i0*z_i0  2*x_i0*y_i0-2*w_i0*z_i0 ...
26     2*w_i0*y_i0+2*x_i0*z_i0 ;
27     2*w_i0*z_i0+2*x_i0*y_i0  w_i0*w_i0-x_i0*x_i0+y_i0*y_i0-z_i0*z_i0 ...
28     2*y_i0*z_i0-2*w_i0*x_i0 ;
29     2*x_i0*z_i0-2*w_i0*y_i0  2*w_i0*x_i0+2*y_i0*z_i0 ...
30     w_i0*w_i0-x_i0*x_i0-y_i0*y_i0+z_i0*z_i0 ];
31
32 M0_IMU=transpose(M0_IMU);
33 %il faut transposer pour avoir wi R i0
34
35 %%--Qualisys--%
36 M0_Q=[data_Q(1,1) data_Q(1,4) data_Q(1,7);
37         data_Q(1,2) data_Q(1,5) data_Q(1,8);
38         data_Q(1,3) data_Q(1,6) data_Q(1,9)];
39 %pas besoin de transposee car on a deja wq R q0
40
41 %% Matrices de rotation courantes %%
42
43 %%--IMU--%
44 M_IMU=zeros(3,3,n-1); %on ne prend pas la 1e valeur car c'est M0_IMU
45
46 for i=2:n
47     w_i=data_IMU(i,4);
48     x_i=data_IMU(i,1);
49     y_i=data_IMU(i,2);
50     z_i=data_IMU(i,3);
51
52     R_IMU = [ w_i*w_i+x_i*x_i-y_i*y_i-z_i*z_i  2*x_i*y_i-2*w_i*z_i ...
53     2*w_i*y_i+2*x_i*z_i ;
54     2*w_i*z_i+2*x_i*y_i  w_i*w_i-x_i*x_i+y_i*y_i-z_i*z_i ...
55     2*y_i*z_i-2*w_i*x_i ;
56     2*x_i*z_i-2*w_i*y_i  2*w_i*x_i+2*y_i*z_i ...
57     w_i*w_i-x_i*x_i-y_i*y_i+z_i*z_i ];
58
59     M_IMU(:,:,i-1)=R_IMU; %i R wi
60 end
61
62 %%--Qualisys--%
63 M_Q=zeros(3,3,m-1); %on ne prend pas la 1e valeur car c'est M0_Q
64
65

```

```

66 for i=2:m
67     R_Q=[data_Q(i,1) data_Q(i,2) data_Q(i,3);
68         data_Q(i,4) data_Q(i,5) data_Q(i,6);
69         data_Q(i,7) data_Q(i,8) data_Q(i,9)];
70         %on prend la transposee pour avoir q R wq
71
72     M_Q(:,:,i-1)=R_Q;
73 end
74
75
76 %% Multiplications des matrices %%
77
78 M_fin_IMU=zeros(3,3,n-1);
79 for i=1:n-1
80     M_fin_IMU(:,:,i)=M_IMU(:,:,i)*M0_IMU(:,:,i);
81 end
82
83 M_fin_Q=zeros(3,3,m-1);
84 for i=1:m-1
85     M_fin_Q(:,:,i)=M_Q(:,:,i)*M0_Q(:,:,i);
86 end
87
88 %% Suppression des donnees en trop %%
89
90 if n<m % si il y a plus de donnees Qualisys
91     M_Q_fin_new=zeros(3,3,n-1);
92     for i=1:n-1
93         M_Q_fin_new(:,:,i)=M_fin_Q(:,:,i);
94     end
95     num_val=n;
96     save("NEW1_4_IMU.mat","M_fin_IMU")
97     save("NEW1_4_Qualisys.mat","M_Q_fin_new")
98
99 else % si il y a plus de donnees IMU
100     M_IMU_fin_new=zeros(3,3,m-1);
101     for i=1:m-1
102         M_IMU_fin_new(:,:,i)=M_fin_IMU(:,:,i);
103     end
104     num_val=m;
105     save("NEW1_4_IMU.mat","M_IMU_fin_new")
106     save("NEW1_4_Qualisys.mat","M_fin_Q")
107 end

```

7.8 Code MATLAB : comparaison matrices de rotation

```

1 clear
2 clc
3 close all
4
5
6 %% Recuperation des matrices de rotation %%
7
8 IMU=load('NEW1_1_IMU.mat');
9 Q=load('NEW1_1_Qualisys.mat');

```

```

10 M_IMU=IMU . M_IMU_fin_new ;
11 M_Q=Q . M_fin_Q ;
12
13
14
15 %% Initialisation %%
16
17 n=length(M_Q) ;
18 ecart=zeros(3 ,3) ;
19 compteur_mauvais=0 ;
20 compteur_bon=0 ;
21
22 esp =0.1; %ecart maximal accepte
23
24
25 %% Comparaison des matrices %%
26
27 for k=1:n
28     for i=1:3
29         for j=1:3
30             ecart(i , j)=abs(M_IMU(i , j , k)-M_Q(i , j , k));
31             if ecart(i , j)>esp
32                 compteur_mauvais=compteur_mauvais +1;
33             else
34                 compteur_bon=compteur_bon +1;
35             end
36         end
37     end
38     ecart;
39 end
40
41 erreur=compteur_mauvais *100/(n*9) %pourcentage de valeurs hors tolerance
42 bon=compteur_bon *100/(n*9) %pourcentage de valeurs considerees égales

```

7.9 Code MATLAB : création des vidéos GIF

```

1 clear all
2 clc
3
4
5 %Chargement des fichiers contenant les données Euler IMU et Qualisys
6
7 RotMatrix_Qualisys=load("NEW1_4_Qualisys.mat");
8 RotMatrix_Qualisys=RotMatrix_Qualisys . M_fin_Q ;
9
10 RotMatrix_IMU=load("NEW1_4_IMU.mat");
11 RotMatrix_IMU=RotMatrix_IMU . M_IMU_fin_new ;
12
13 % Coordonnées des différents points A à H de la modélisation du phidget
14
15 A_Q = [0 ; 0 ; 0];
16 B_Q = [1 ; 0 ; 0];
17 C_Q = [1 ; 1 ; 0];
18 D_Q = [0 ; 1 ; 0];

```

```

19 E_Q = [ 0 ; 0 ; 1 ];
20 F_Q = [ 1 ; 0 ; 1 ];
21 G_Q = [ 1 ; 1 ; 1 ];
22 H_Q = [ 0 ; 1 ; 1 ];
23
24 A_IMU = [ 0 ; 0 ; 0 ];
25 B_IMU = [ 1 ; 0 ; 0 ];
26 C_IMU = [ 1 ; 1 ; 0 ];
27 D_IMU = [ 0 ; 1 ; 0 ];
28 E_IMU = [ 0 ; 0 ; 1 ];
29 F_IMU = [ 1 ; 0 ; 1 ];
30 G_IMU = [ 1 ; 1 ; 1 ];
31 H_IMU = [ 0 ; 1 ; 1 ];
32
33 At_Q=transpose(A_Q);
34 Bt_Q=transpose(B_Q);
35 Ct_Q=transpose(C_Q);
36 Dt_Q=transpose(D_Q);
37 Et_Q=transpose(E_Q);
38 Ft_Q=transpose(F_Q);
39 Gt_Q=transpose(G_Q);
40 Ht_Q=transpose(H_Q);
41
42 At_IMU=transpose(A_IMU);
43 Bt_IMU=transpose(B_IMU);
44 Ct_IMU=transpose(C_IMU);
45 Dt_IMU=transpose(D_IMU);
46 Et_IMU=transpose(E_IMU);
47 Ft_IMU=transpose(F_IMU);
48 Gt_IMU=transpose(G_IMU);
49 Ht_IMU=transpose(H_IMU);
50
51 % Matrices contenant les points et les numeros des points permettant
52 % de construire chaque face
53
54 Points_IMU = [At_IMU;Bt_IMU;Ct_IMU;Dt_IMU;Et_IMU;Ft_IMU;Gt_IMU;Ht_IMU];
55 Points_Qualisys=[At_Q;Bt_Q;Ct_Q;Dt_Q;Et_Q;Ft_Q;Gt_Q;Ht_Q];
56 Faces_IMU = [1 2 6 5; 2 3 7 6; 3 4 8 7; 4 1 5 8 ; 1 2 3 4; 5 6 7 8];
57 Faces_Qualisys = [1 2 6 5; 2 3 7 6; 3 4 8 7; 4 1 5 8 ; 1 2 3 4; 5 6 7 8];
58
59
60 nbIm = min(length(RotMatrix_IMU),length(RotMatrix_Quaternion));
61
62 %%%%%% Calcul pour toutes les matrices de rotations dont on dispose %%%%%%
63
64 for i =1:nbIm
65
66     idx=int2str(i);
67     MatRot_Qualisys=RotMatrix_Qualisys (:,:,i);
68     MatRot_IMU=RotMatrix_IMU (:,:,i);
69
70 %%%%%% Calcul de la rotation au nouvel instant du pdv Qualisys %%%%%%
71
72     A_Q=MatRot_Qualisys*A_Q;
73     B_Q=MatRot_Qualisys*B_Q;
74     C_Q=MatRot_Qualisys*C_Q;

```

```

75 D_Q=MatRot_Qualisys*D_Q;
76 E_Q=MatRot_Qualisys*E_Q;
77 F_Q=MatRot_Qualisys*F_Q;
78 G_Q=MatRot_Qualisys*G_Q;
79 H_Q=MatRot_Qualisys*H_Q;

80
81 At_Q=transpose(A_Q);
82 Bt_Q=transpose(B_Q);
83 Ct_Q=transpose(C_Q);
84 Dt_Q=transpose(D_Q);
85 Et_Q=transpose(E_Q);
86 Ft_Q=transpose(F_Q);
87 Gt_Q=transpose(G_Q);
88 Ht_Q=transpose(H_Q);

89
90 Points_Qualisys =[At_Q;Bt_Q;Ct_Q;Dt_Q;Et_Q;Ft_Q;Gt_Q;Ht_Q];
91
92 %%%%%%%% Calcul de la rotation au nouvel instant du pdv Phidget %%%%%%
93
94 A_IMU=MatRot_IMU*A_IMU;
95 B_IMU=MatRot_IMU*B_IMU;
96 C_IMU=MatRot_IMU*C_IMU;
97 D_IMU=MatRot_IMU*D_IMU;
98 E_IMU=MatRot_IMU*E_IMU;
99 F_IMU=MatRot_IMU*F_IMU;
100 G_IMU=MatRot_IMU*G_IMU;
101 H_IMU=MatRot_IMU*H_IMU;

102
103 At_IMU=transpose(A_IMU);
104 Bt_IMU=transpose(B_IMU);
105 Ct_IMU=transpose(C_IMU);
106 Dt_IMU=transpose(D_IMU);
107 Et_IMU=transpose(E_IMU);
108 Ft_IMU=transpose(F_IMU);
109 Gt_IMU=transpose(G_IMU);
110 Ht_IMU=transpose(H_IMU);

111
112 Points_IMU =[At_IMU;Bt_IMU;Ct_IMU;Dt_IMU;Et_IMU;Ft_IMU;Gt_IMU;Ht_IMU];
113
114 %%%%%%%% Deplacement du phidget du point de vue Phidget %%%%%%
115
116 f_IMU=figure();
117 plot3(Points_IMU(:,1),Points_IMU(:,2),Points_IMU(:,3),'black')
118 hold on
119 scatter3(Points_IMU(1,1),Points_IMU(1,2),Points_IMU(1,3),...
120 'black','filled')
121 hold on
122 scatter3(Points_IMU(4,1),Points_IMU(4,2),Points_IMU(4,3),'b','filled')
123 hold on
124 scatter3(Points_IMU(2,1),Points_IMU(2,2),Points_IMU(2,3),'g','filled')
125 hold on

126
127 patch('Vertices',Points_IMU,'Faces',Faces_IMU,...
128 'FaceVertexCData',hsv(6),'FaceColor','flat')
129 hold off
130

```

```

131 axis([-2 2 -2 2 -2 2])
132 xlabel('x', 'FontSize', 10)
133 ylabel('y', 'FontSize', 10)
134 zlabel('z', 'FontSize', 10)
135 grid on
136
137 saveas(f_IMU, "imageIMU3_" + idx, 'png')
138
139 close(f_IMU)
140
141 %%%%%% Deplacement du phidget du point de vue Qualisys %%%%%%
142
143 f_Qualisys=figure();
144 plot3(Points_Qualisys(:,1), Points_Qualisys(:,2), ...
145 Points_Qualisys(:,3), 'black')
146 hold on
147 scatter3(Points_Qualisys(1,1), Points_Qualisys(1,2), ...
148 Points_Qualisys(1,3), 'black', 'filled')
149 hold on
150 scatter3(Points_Qualisys(4,1), Points_Qualisys(4,2), ...
151 Points_Qualisys(4,3), 'b', 'filled')
152 hold on
153 scatter3(Points_Qualisys(2,1), Points_Qualisys(2,2), ...
154 Points_Qualisys(2,3), 'g', 'filled')
155 hold on
156
157 patch('Vertices', Points_Qualisys, 'Faces', Faces_Qualisys, ...
158 'FaceVertexCData', hsv(6), 'FaceColor', 'flat')
159 hold off
160
161 axis([-2 2 -2 2 -2 2])
162 xlabel('x', 'FontSize', 10)
163 ylabel('y', 'FontSize', 10)
164 zlabel('z', 'FontSize', 10)
165 grid on
166
167 saveas(f_Qualisys, "imageQualisys3_" + idx, 'png')
168
169 close(f_Qualisys)
170
171 end
172
173 %%%%%% Creation de la video %%%%%%
174
175 filename = 'Video_4_3.gif';
176
177
178 f = figure();
179 for idx = 1:nbIm
180
181
182 im = imread("imageIMU3_" + idx + ".png");
183 subplot(121)
184 imshow(im)
185 title("IMU")
186

```

```
187 im = imread ("imageQualisys3_" + idx + ".png");
188 subplot(122)
189 imshow(im)
190 title("Qualisys")
191
192 frame = getframe(f);
193 frame = frame2im(frame);
194
195 pause(1)
196
197 [A, map] = rgb2ind(frame, 256);
198 if idx == 1
199     imwrite(A, map, filename, 'gif', 'LoopCount', Inf, 'DelayTime', 1);
200     imwrite(A, map, filename, 'gif', 'WriteMode', 'append', 'DelayTime', 1);
201 else
202     imwrite(A, map, filename, 'gif', 'WriteMode', 'append', 'DelayTime', 1);
203 end
204
205
206 %%%%%%%%%%%%%%
```

Références

- [1] Quaternions
<https://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions>
- [2] Geometry Rotations Euler
<https://www.euclideanspace.com/maths/geometry/rotations/euler/index.htm>
- [3] Conversion Euler To Quaternions
www.euclideanspace.com/maths/geometry/rotations/conversions/eulerToQuaternion/index.htm
- [4] Quaternions et rotation dans l'espace
https://fr.wikipedia.org/wiki/Quaternions_et_rotation_dans_l'espace
- [5] Centrale à inertie
https://fr.wikipedia.org/wiki/Centrale_à_inertie
- [6] A Guide To using IMU
http://www.starlino.com/imu_guide.html
- [7] La centrale inertielles
<https://www.cadden.fr/centrale-inertielles-fonctionnement/>
- [8] API Documentation at Phidgets
<https://www.Phidgets.com/?view=api/>
- [9] *QTM Marine Manual : User manual*, Qualisys AB, 2008
<https://usermanual.wiki/Document/QTM20Marine20manual2020081211.591357325>
- [10] Caméras MRI Qualisys
<https://www.qualisys.com/cameras/oqus-mri/>