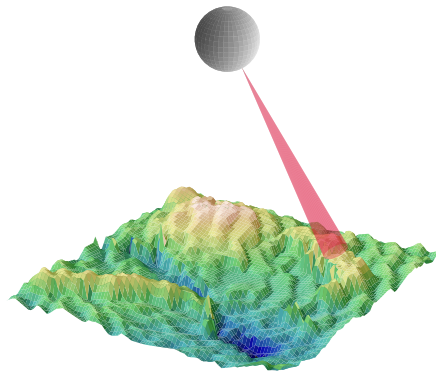

Cartographie et caractérisation des fonds marins par un sonar monofaisceau orienté par une sphère

Samuel Lerman



TIPE, thème Océan
Commencé en 2020, terminé en 2026

Abstract

A single-beam sonar measures one range per ping; to map a sea floor, its beam must be swept. In this work, the transducer is fixed in a sealed sphere, fully immersed, that also carries the electronics and an inertial measurement unit. Three omnidirectional wheels pressing on its upper half turn it about any axis; the net buoyancy of the sphere keeps it against them. Every beam axis is a radius of the sphere, so all beams start from its fixed centre : the system has no parallax and no singular direction, and the water exerts no torque on it. When the contact points lie $35,26^\circ$ from the pole, the three wheel axles are orthogonal and the drive is isotropic. The physics of the system is described (propagation, transducer directivity, backscatter, sonar equation, statics and kinematics of the sphere) and simulated : relief generation, environment, echo synthesis with speckle, orientation estimation and control. In simulation, the control program points the beam with a mean error of $0,22^\circ$ over a 166 s scan of a 50×50 cm model seabed. Classical bottom detection maps it with an RMS error of 9,8 mm ; a Richardson-Lucy deconvolution of the angle-range data cube reduces it to 7,5 mm. Averaged over homogeneous areas, the angular dependence of backscatter, measured from five positions of the sphere, separates four seabed types ; cell by cell, the classification is right in 64% of cases on average over the four types. The firmware of the two boards and the acquisition and imaging programs that would drive the real system are given, with an experimental protocol. The system has not been built : all results are simulated.

Introduction

Ce document présente un travail commencé en 2020 dans le cadre de l'épreuve de TIPE à préparer pour les concours des classes préparatoires aux grandes écoles (CPGE) scientifiques, sur le thème « Océan ». Je n'avais pas pu le mener à son terme ; j'ai décidé de m'y replonger en 2026 pour le finir.

Le sujet

Un sonar est un appareil qui « voit » avec le son. Il émet une brève impulsion sonore et écoute son écho : le temps que met l'écho à revenir donne la distance de l'obstacle, comme pour un écho en montagne. Sous l'eau, où la lumière porte mal mais où le son se propage loin, c'est le moyen habituel de mesurer la profondeur et de cartographier le fond. Un sonar monofaisceau n'émet que dans une seule direction, dans un cône étroit : chaque impulsion ne donne la distance que d'une petite tache du fond. Pour obtenir une image, il faut donc orienter le faisceau successivement dans de nombreuses directions, c'est-à-dire le balayer. Nous proposons pour cela de fixer l'élément qui émet et reçoit le son, le transducteur, dans une sphère immergée que l'on fait tourner sur elle-même, comme un œil dans son orbite.

Démarche

Le dispositif n'a pas été construit. Nous avons donc procédé de la façon suivante :

1. étudier la physique de la mesure pour choisir le transducteur et comprendre ce que mesure réellement un sonar monofaisceau ;
2. concevoir le dispositif d'orientation et en étudier la mécanique : l'équilibre de la sphère dans l'eau, son entraînement, la mesure et l'asservissement de son orientation ;
3. écrire un simulateur qui, à partir d'un fond connu, produit les données que fournirait le dispositif réel, avec ses imperfections ;
4. écrire les programmes de pilotage et d'imagerie qui serviraient tels quels sur le dispositif réel, et les faire fonctionner sur ces données simulées ;
5. comparer le fond reconstruit au fond connu, ce qui mesure la fidélité de l'image, puis caractériser la nature du fond ;
6. préparer la réalisation : programmes des cartes électroniques et protocole expérimental.

Tous les résultats présentés sont donc issus de simulations ; ce qui reste à vérifier par l'expérience est résumé au chapitre 7.

Plan

Dans une première partie, nous présentons la « Mise en Cohérence des Objectifs du TIPE » (MCOT), qui permet de contextualiser le sujet et de définir la problématique retenue. Le chapitre 2 étudie la physique de la mesure. Le chapitre 3 est consacré à la sphère : son équilibre dans l'eau, son entraînement, la mesure et l'asservissement de son orientation. Le chapitre 4 présente la simulation et ses résultats : création des reliefs, modélisation de l'environnement, acquisition, reconstruction et caractérisation du fond. Le chapitre 5 décrit les programmes qui piloteraient le dispositif réel et propose un protocole expérimental. Le chapitre 6 transpose le dispositif en mer, et le chapitre 7 fait le bilan de ce qui a été fait, des limites de l'étude et de ce qu'il reste à faire.

Table des matières

Abstract	1
Introduction	1
1 Mise en cohérence des objectifs du TIPE	4
Problématique	5
Objectifs	5
Lien avec le thème « Océan » et motivations	5
Positionnement thématique	5
Mots clés	5
Bibliographie commentée	5
2 Le sonar monofaisceau	7
2.1 Propagation des ondes acoustiques dans l'eau	7
2.2 Le transducteur	8
2.3 Résolution spatiale	9
2.4 Écho renvoyé par le fond	10
2.5 Mesure de la profondeur	11
3 Orientation du faisceau par une sphère	13
3.1 Principe du dispositif	13
3.2 Équilibre de la sphère dans l'eau	15
3.3 Entraînement par trois roues	16
3.4 Orientation de la sphère et pointage du faisceau	17
3.5 Mesure de l'orientation	18
3.6 Asservissement de l'orientation	20
4 Simulation de l'acquisition et imagerie du fond	22
4.1 Organisation des programmes	22
4.2 Création des reliefs	22
4.3 Modélisation de l'environnement	23
4.4 Simulation de l'acquisition	23
4.5 Reconstruction du relief	24
4.5.1 Méthode classique : détection du fond	24
4.5.2 Méthode proposée : déconvolution des données	24
4.5.3 Résultats	25
4.5.4 Influence des paramètres	27
4.6 Caractérisation des fonds	29
5 Réalisation du système	31
5.1 Architecture	31
5.2 Carte du berceau	31
5.3 Sonde	32
5.4 Chaîne d'émission et de réception	32
5.5 Programmes d'acquisition et d'imagerie	32
5.6 Protocole expérimental proposé	33

6	Transposition en mer	34
6.1	Similitude	34
6.2	Choix de la fréquence	34
6.3	Stabilisation du faisceau	35
6.4	Tenue à la pression	35
6.5	Réfraction	35
7	Bilan, limites et suite du travail	36
7.1	Ce qui a été fait	36
7.2	Ce qui n'a pas été fait	36
7.3	Limites de l'étude	36
7.4	Ce qu'il reste à faire	37
	Conclusion	38
	Bibliographie	39
	Annexes	40
	Annexe 1 : simulation de l'écho	40
	Annexe 2 : imagerie	42
	Annexe 3 : la sphère	45
	Annexe 4 : mesure et asservissement de l'orientation	48
	Annexe 5 : carte du berceau	52
	Annexe 6 : sonde	55

Chapitre 1

Mise en cohérence des objectifs du TIPE

Dans ce chapitre, nous posons le cadre de travail dans le formalisme attendu pour le TIPE. Cette partie sert essentiellement à contextualiser le sujet et à expliquer le fonctionnement d'un sonar dans les grandes lignes. Nous présentons la problématique retenue ainsi que les objectifs que nous nous sommes fixés. Nous expliquons en quoi ce sujet s'inscrit dans le thème Océan et nos motivations. Nous présentons ensuite le positionnement thématique et quelques mots clés décrivant ce travail. Enfin, nous exposons la bibliographie commentée, qui correspond à un état de l'art.

Problématique

Comment, par quels moyens restituer le plus fidèlement possible une image acoustique des fonds marins à partir de données reçues par un sonar monofaisceau, de façon à pouvoir en caractériser la morphologie ?

Objectifs

On se propose de réaliser les principaux points ci-dessous :

- Étudier les mécanismes d'acquisition des données, la propagation des ondes acoustiques en milieu marin et leur interaction avec le fond.
- Mettre au point un système de type sonar monofaisceau et une maquette expérimentale ayant les mêmes propriétés que le milieu marin, permettant de restituer une image précise des fonds.
- Caractériser le fond par l'étude de bilans énergétiques et de la réverbération angulaire.

Lien avec le thème « Océan » et motivations

Lien avec le thème

La caractérisation des fonds marins est particulièrement importante pour les entreprises spécialisées dans les aménagements hydrauliques et la sécurisation des biens ou des personnes. La cartographie des fonds, en 2D ou en 3D, offre diverses applications : l'étude des terrains et de leur évolution, des données cartographiques pour la protection des milieux sensibles, la détection d'épaves ou d'objets obstruants, le suivi et la maintenance d'installations, comme les conduites sous-marines, par drone autonome.

Motivations

Connaître la topographie des fonds marins est essentiel à la prévention et au suivi de l'activité océanique : c'est ce qui nous a motivés à étudier ce sujet.

Positionnement thématique

Acoustique, Modélisation informatique, Électronique analogique, Analyse spectrale

Mots clés

Mots clés (en français)

Transducteur électroacoustique
Onde acoustique
Bruit
Zone insonifiée
Données cartographiques

Mots clés (en anglais)

Electroacoustic transducer
Acoustic wave
Noise
Insonified area
Geographic positioning data

Bibliographie commentée

Un sonar actif émet une impulsion acoustique et écoute ses échos. L'ouvrage de Lurton [1] en donne les bases pour les applications sous-marines : propagation, transducteurs, équation du sonar, interaction avec le fond, et les grandes familles de sondeurs. Le sondeur monofaisceau, le plus simple, mesure une seule profondeur par impulsion, celle de la tache insonifiée sous le navire ; le sondeur multifaisceau forme électroniquement des centaines de faisceaux en éventail et cartographie une bande entière à chaque impulsion. Les petits robots sous-marins utilisent aussi des sonars dont la tête est orientée mécaniquement [4].

Le transducteur se modélise comme un piston plan circulaire, dont la directivité est la transformée de Fourier de l'ouverture [3, 2] : elle fixe la résolution latérale, tandis que la durée de l'impulsion fixe la résolution axiale. La célérité du son [5] et son absorption [6] dépendent de la température, de la salinité et de la fréquence ; l'absorption impose le compromis entre portée et résolution. L'écho du fond dépend de son indice de rétrodiffusion, qui varie avec l'angle d'incidence et la nature du sédiment : réflexion quasi spéculaire autour de la verticale, diffusion selon la loi de Lambert aux incidences obliques [11, 9, 10]. Les méthodes de classification des fonds reposent sur cette réponse angulaire. Enfin, l'écho est la somme des ondes renvoyées par un grand nombre de diffuseurs : son

amplitude fluctue de façon aléatoire, c'est le chatoiement (*speckle*) [12].

Pour orienter le capteur, nous nous inspirons des robots en équilibre sur une boule entraînée par des roues [14, 15]. L'orientation se représente par des quaternions [16], s'estime par un filtre complémentaire [17] et s'asservit par un retour sur l'erreur d'orientation [19]. Pour reconstruire le fond à partir d'échos flous, nous utilisons l'algorithme de déconvolution de Richardson-Lucy [21, 22], conçu pour les images astronomiques.

Chapitre 2

Le sonar monofaisceau

Ce chapitre présente l'aspect théorique de la mesure. Nous décrivons dans un premier temps la propagation des ondes acoustiques dans l'eau, puis le transducteur et la résolution spatiale qu'il permet d'atteindre. Nous étudions ensuite l'écho renvoyé par le fond, son niveau et ses fluctuations, ce qui nous permet de préciser ce que mesure réellement un sonar monofaisceau.

Principe de la mesure

Le transducteur est un disque piézoélectrique : soumis à une tension alternative, il vibre et émet une onde sonore ; frappé par une onde, il produit une tension. Il sert donc à la fois d'émetteur et de récepteur. Il émet une impulsion de quelques microsecondes, puis enregistre l'écho. Si l'écho revient au bout du temps t , l'obstacle est à la distance

$$r = \frac{ct}{2} \quad (2.1)$$

où c est la célérité du son dans l'eau, le facteur 2 venant de l'aller-retour. Les grandeurs acoustiques varient sur plusieurs ordres de grandeur : on les exprime en décibels, $10 \log(I/I_0)$ pour un rapport d'intensités, de sorte qu'une perte de 10 dB divise l'intensité par 10 et une perte de 20 dB par 100.

2.1 Propagation des ondes acoustiques dans l'eau

Célérité du son

La célérité du son dans l'eau dépend de la température T (°C), de la salinité S (g/kg) et de la profondeur z (m). On utilise la formule de Medwin [5] :

$$c = 1449,2 + 4,6T - 0,055T^2 + 0,00029T^3 + (1,34 - 0,010T)(S - 35) + 0,016z \quad (2.2)$$

Dans l'eau douce du bassin, à 20 °C, on obtient $c = 1482$ m/s. La célérité augmente de 3,1 m/s par degré : une erreur d'un degré sur la température fausse une distance de 40 cm de 0,83 mm. Il faut donc mesurer la température de l'eau. Dans le bassin, la température est presque uniforme et les rayons sonores restent rectilignes.

Pertes par propagation

L'intensité d'une onde sphérique décroît comme $1/r^2$ et l'eau en absorbe une partie. La perte par propagation sur un trajet r vaut, en décibels :

$$TL = 20 \log r + \alpha r \quad (2.3)$$

Le coefficient d'absorption α est donné par la formule de Francois et Garrison [6], somme de trois termes : l'acide borique et le sulfate de magnésium, tous deux proportionnels à la salinité et donc absents de l'eau douce, et la viscosité de l'eau, qui croît comme le carré de la fréquence (Fig 2.1). Dans le bassin, à 1 MHz, $\alpha = 0,22$ dB/m : l'absorption est négligeable sur un mètre. En mer, α vaut 3,3 dB/km à 20 kHz, 54 dB/km à 200 kHz et 0,36 dB/m à 1 MHz : plus la fréquence est élevée, plus la portée est courte.

Réverbération dans le bassin

Dans un bassin, le son se réfléchit sur les parois, le fond et la surface libre. Il faut attendre que cette réverbération s'éteigne avant d'émettre l'impulsion suivante. En traitant le bassin comme une salle, la formule d'Eyring [7] donne la durée de décroissance de 60 dB :

$$T_{60} = \frac{24 \ln 10 V}{-c S \ln(1 - \bar{a})} \quad (2.4)$$

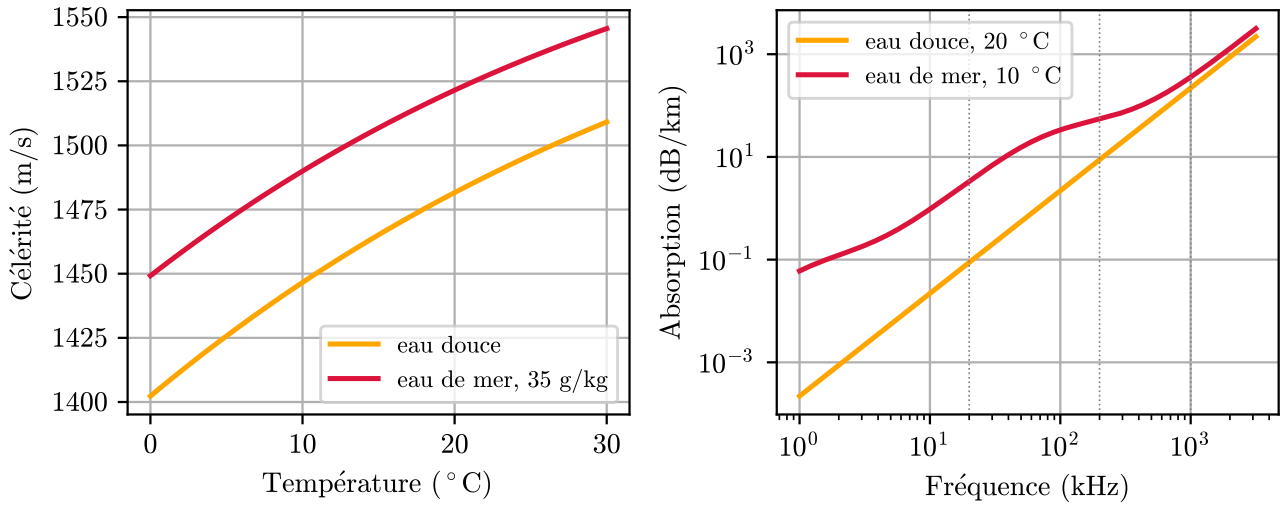


FIGURE 2.1 – Célérité du son en fonction de la température (à gauche) et absorption en fonction de la fréquence (à droite)

où V est le volume d'eau, S la surface des parois et $\bar{a}$ leur coefficient d'absorption moyen ($1 - R^2$ pour une paroi de coefficient de réflexion R , soit 0,34 pour le verre et 0,87 pour le sable ; la surface libre réfléchit tout). Pour un bassin vitré de 80×80 cm rempli sur 55 cm, on trouve $T_{60} \approx 9,0$ ms. Avec une impulsion toutes les 20 ms, soit 50 par seconde, la réverbération est éteinte avant l'émission suivante.

2.2 Le transducteur

Fonction de directivité

La directivité du transducteur exprime la distribution angulaire de l'énergie acoustique qu'il émet à une fréquence f donnée. Tous les points du disque vibrent ensemble : dans l'axe, leurs ondes arrivent en phase et s'additionnent ; sur les côtés, elles arrivent avec des retards différents et se compensent en partie. Le son est donc concentré dans un cône autour de l'axe, le faisceau. Mathématiquement, la fonction de directivité est la transformée de Fourier de l'ouverture du transducteur : elle dépend donc de sa géométrie [1]. Pour un piston circulaire plan de rayon a , elle vaut, en amplitude [2] :

$$D(\theta) = \frac{2J_1(ka \sin \theta)}{ka \sin \theta} \quad (2.5)$$

où J_1 est la fonction de Bessel de première espèce d'ordre 1 et $k = 2\pi f/c$ le nombre d'onde. Elle est maximale dans l'axe, puis s'annule pour $ka \sin \theta = 3,83$ (Fig 2.2). Le rayonnement présente donc un lobe principal, caractérisé par son ouverture angulaire à -3 dB notée $2\theta_0$, et des lobes secondaires, le premier à $-17,6$ dB, qui peuvent être à l'origine d'erreurs d'interprétation.

Ouverture du faisceau

La largeur du lobe principal à -3 dB correspond à $2ka \sin \theta_0 \approx 3,2$, ce qui donne, pour de petits angles :

$$2\theta_0 \approx 2 \sin \theta_0 \approx \frac{3,2}{ka} \approx \frac{\lambda}{2a} \quad (2.6)$$

Un transducteur de rayon a est d'autant plus directif que sa fréquence est élevée. Le même transducteur sert à l'émission et à la réception : l'écho est pondéré par D^2 en amplitude et le faisceau aller-retour est plus étroit, avec $2ka \sin \theta_0 \approx 2,3$ à -3 dB.

Choix du transducteur

Le fond de la maquette se trouve à 30 à 55 cm du transducteur et ses détails mesurent quelques centimètres. Il faut donc un faisceau étroit, c'est-à-dire une fréquence élevée, ce que permet la faible absorption dans le bassin. Nous retenons un disque piézoélectrique de 12,7 mm de diamètre ($a = 6,35$ mm) résonnant vers 1 MHz, du type de ceux utilisés en contrôle non destructif. Dans l'eau à 20 °C, $\lambda = 1,48$ mm et $ka = 26,9$. L'ouverture à -3 dB vaut alors $6,9^\circ$ à l'émission et $4,9^\circ$ en aller-retour, le premier zéro est à $8,2^\circ$ de l'axe et l'indice de directivité, $10 \log(ka)^2$, vaut 28,6 dB.

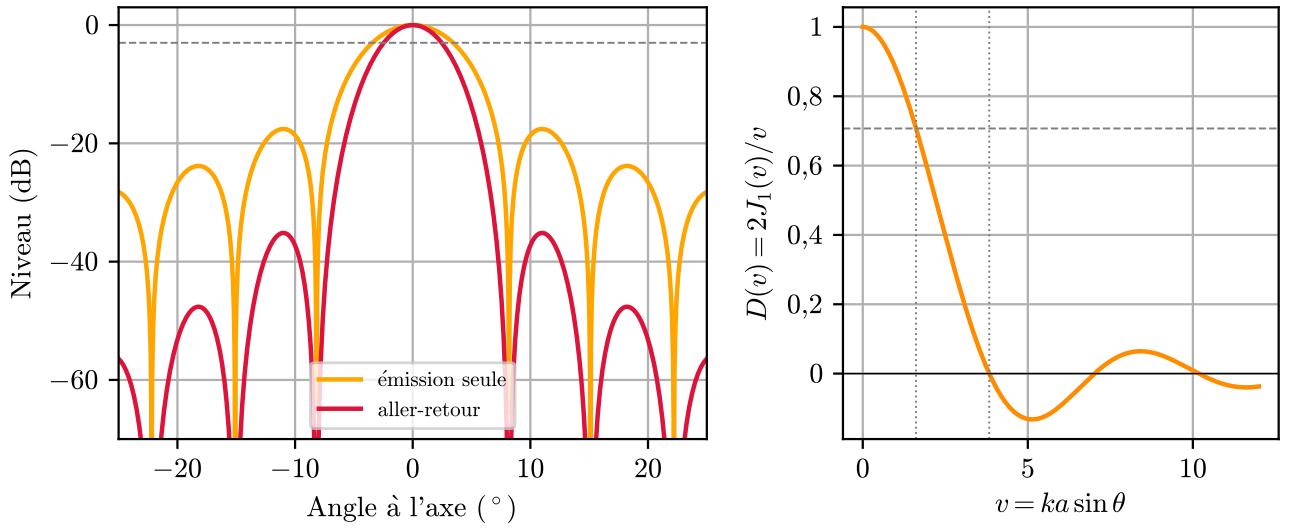


FIGURE 2.2 – Directivité du disque retenu ($ka = 26,9$) à l'émission et en aller-retour (à gauche), et fonction de directivité du piston circulaire (à droite)

Champ proche et champ lointain

Près du transducteur, les ondes issues des différents points du disque interfèrent : la pression dans l'axe oscille et la directivité (2.5) ne s'applique pas. Dans l'axe d'un piston, la pression vaut [3] :

$$p(z) \propto \left| \sin \left(\frac{\pi}{\lambda} \left(\sqrt{z^2 + a^2} - z \right) \right) \right| \quad (2.7)$$

Son dernier maximum est à $a^2/\lambda = 2,7$ cm. Au-delà de la distance de Rayleigh, $\pi a^2/\lambda = 8,5$ cm, la pression décroît comme $1/z$: c'est le champ lointain (Fig 2.3). Le fond, à plus de 30 cm, s'y trouve.

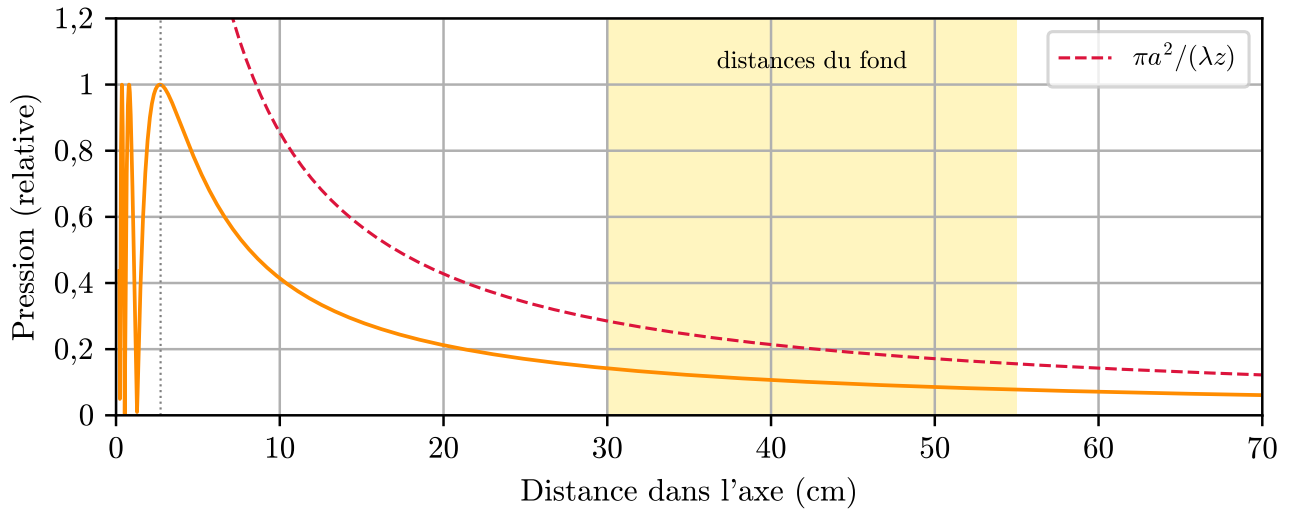


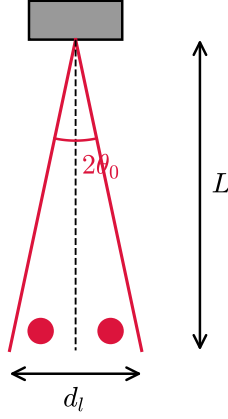
FIGURE 2.3 – Pression dans l'axe du transducteur : champ proche et champ lointain

2.3 Résolution spatiale

La résolution spatiale d'un système est sa capacité à séparer deux cibles proches : c'est la distance minimale qui doit les séparer pour qu'elles soient discernables l'une de l'autre. Il en existe deux sortes (Fig 2.4) :

- la résolution latérale d_l , qui est l'aptitude à différencier deux cibles ponctuelles dans une direction perpendiculaire à l'axe de propagation ;
- la résolution axiale d_a , qui caractérise l'aptitude à différencier deux cibles ponctuelles situées dans l'axe de propagation.

Résolution latérale



Résolution axiale : deux échos

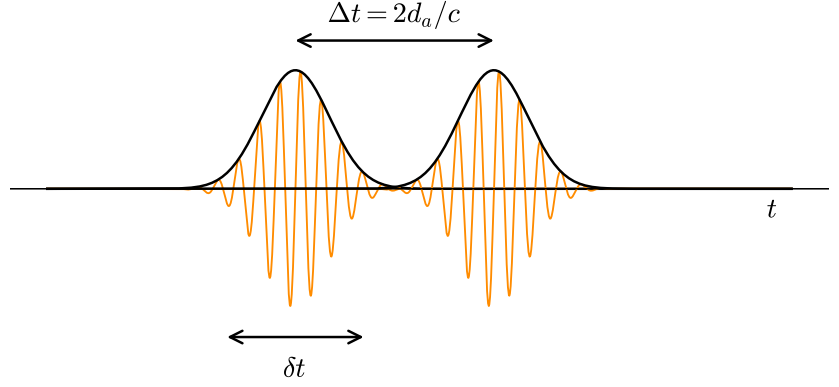


FIGURE 2.4 – Résolution latérale d_l et résolution axiale d_a du transducteur

Résolution latérale

À la distance L , deux cibles ne sont séparées que si elles ne se trouvent pas dans le faisceau en même temps. Dans l'approximation des petits angles :

$$d_l = 2L \tan \theta_0 \approx 2L\theta_0 \approx \frac{\lambda L}{2a} = \frac{cL}{2af} \quad (2.8)$$

La résolution latérale se dégrade donc avec la distance. Pour notre transducteur, à 35 cm, la tache insonifiée mesure 4,2 cm à -3 dB à l'émission et 3,0 cm en aller-retour.

Résolution axiale

Deux cibles placées dans l'axe, distantes de d_a , renvoient des échos séparés de $\Delta t = 2d_a/c$. Pour les distinguer, il faut que cet intervalle soit supérieur à la durée δt de l'impulsion, d'où :

$$d_a = \frac{c \delta t}{2} \quad (2.9)$$

Avec 5 périodes à 1 MHz, $\delta t = 5 \mu s$ et $d_a = 3,7$ mm. L'enveloppe de l'écho est échantillonnée à 500 kHz, soit un échantillon tous les 1,48 mm.

Cas d'un faisceau incliné

Quand le faisceau est incliné d'un angle θ par rapport à la normale au fond, la tache s'allonge et l'impulsion ne l'éclaire plus d'un coup : l'onde balaie le fond à la vitesse $c/(2 \sin \theta)$ et découpe la tache en bandes de largeur $c \delta t / (2 \sin \theta)$, soit 7 mm à 30° . À la verticale, la résolution est fixée par le faisceau ; de biais, elle l'est en partie par l'impulsion.

2.4 Écho renvoyé par le fond

Indice de rétrodiffusion

Le fond ne renvoie vers le transducteur qu'une petite partie de l'énergie qu'il reçoit ; cette partie dépend de sa nature et de l'angle d'incidence θ , angle entre le faisceau et la perpendiculaire au fond. Une petite surface dA du fond, éclairée sous l'incidence θ par une onde d'intensité I_i , renvoie vers le transducteur, à la distance r , l'intensité :

$$I_s = I_i \frac{\sigma(\theta) dA}{r^2} \quad (2.10)$$

où σ est la section de rétrodiffusion par unité d'aire et d'angle solide ; $BS = 10 \log \sigma$ est l'indice de rétrodiffusion. Il résulte de deux mécanismes [9, 10] :

- près de la verticale, une réflexion quasi spéculaire sur les facettes de la rugosité qui font face au transducteur. Si les pentes de la rugosité suivent une loi gaussienne d'écart type δ dans chaque direction, l'approximation de Kirchhoff donne :

$$\sigma_K(\theta) = \frac{R_0^2}{8\pi\delta^2 \cos^4 \theta} \exp\left(-\frac{\tan^2 \theta}{2\delta^2}\right) \quad (2.11)$$

où $R_0 = (Z - 1)/(Z + 1)$ est le coefficient de réflexion en incidence normale, Z étant l'impédance du sédiment (ρc) rapportée à celle de l'eau ;

— aux incidences obliques, une diffusion par les petites irrégularités du sédiment, décrite par la loi de Lambert [11] : $\sigma_L(\theta) = \mu \cos^2 \theta$.

On prend $\sigma = \sigma_K + \sigma_L$, avec des valeurs typiques pour quatre fonds (Fig 2.5). Ces valeurs reproduisent les ordres de grandeur publiés [1, 9, 10] ; ce ne sont pas des mesures. On remarque qu'à la verticale tous les fonds renvoient beaucoup d'énergie, alors que de biais la vase renvoie 20 à 30 dB de moins que la roche : c'est la réponse angulaire qui permet de distinguer les fonds.

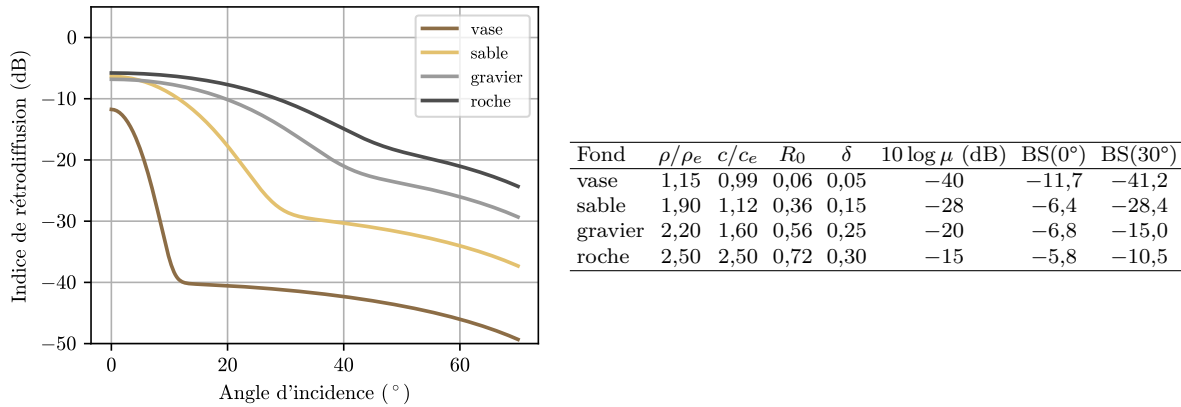


FIGURE 2.5 – Indice de rétrodiffusion des quatre fonds en fonction de l'angle d'incidence, et leurs paramètres

Équation du sonar

En sommant sur la tache insonifiée, on obtient le niveau de l'écho, en dB re 1 μ Pa [8, 1] :

$$EL = SL - 2TL + BS(\theta) + 10 \log A, \quad A \approx \frac{r^2 \Psi}{\cos \theta} \quad (2.12)$$

où SL est le niveau d'émission à 1 m et A l'aire de la tache, exprimée à l'aide de l'angle solide équivalent du faisceau aller-retour $\Psi = \int D^4 d\Omega$ [13] ($10 \log \Psi = -21,0$ dB pour notre disque). Pour un petit transducteur excité sous quelques dizaines de volts, nous prenons $SL = 200$ dB re 1 μ Pa à 1 m et une sensibilité en réception de -200 dB re 1 V/ μ Pa, ordres de grandeur qu'il faudra mesurer. Le sable à la verticale renvoie alors 182 dB, soit 121 mV aux bornes du transducteur ; la vase vue sous 38° ne renvoie que 145 dB, soit 1,69 mV. Le bruit d'un bon préamplificateur, ramené à l'entrée, est de l'ordre de 1,3 μ V : le rapport signal sur bruit est confortable, mais la dynamique des échos atteint 37 dB. C'est pourquoi la chaîne de réception utilise un détecteur logarithmique (section 5.4).

Fluctuations de l'écho

L'écho est la somme des ondes renvoyées par un très grand nombre de petits diffuseurs, dont les phases $2kr$ varient de 2π quand r varie d'une demi-longueur d'onde, soit 0,7 mm. Ces phases sont pratiquement aléatoires : d'après le théorème central limite, l'amplitude de l'écho suit une loi de Rayleigh et son intensité une loi exponentielle, dont l'écart type est égal à la moyenne [12]. L'intensité d'une impulsion fluctue donc de plusieurs décibels autour de sa moyenne : c'est le chatoiement. Pour une géométrie fixe, ces fluctuations sont figées : répéter l'impulsion ne les réduit pas, il faut moyenner des mesures faites sous des angles différents.

2.5 Mesure de la profondeur

Principe

Le sonar ne mesure pas la distance d'un point mais l'écho de toute une tache (Fig 2.7). Pour en tirer une profondeur, on peut retenir :

- le front montant de l'écho, c'est-à-dire le point le plus proche de la tache. C'est le choix des sondeurs de navigation, qui cherchent la profondeur minimale ;
- le barycentre de l'énergie de l'écho, plus proche de la distance dans l'axe quand le fond est régulier.

Limites

Sur un fond plat, à la verticale, les deux mesures coïncident à peu près (Fig 2.6). De biais, l'écho s'étale sur toute la tache. Au bord d'un objet, la tache contient à la fois l'objet et le fond : l'écho présente deux maxima et aucune des deux mesures ne donne la profondeur dans l'axe. Une image construite en reportant chaque mesure dans l'axe du faisceau est donc floue à l'échelle de la tache. Le chapitre 4 propose une méthode qui exploite le recouvrement des taches voisines pour faire mieux.

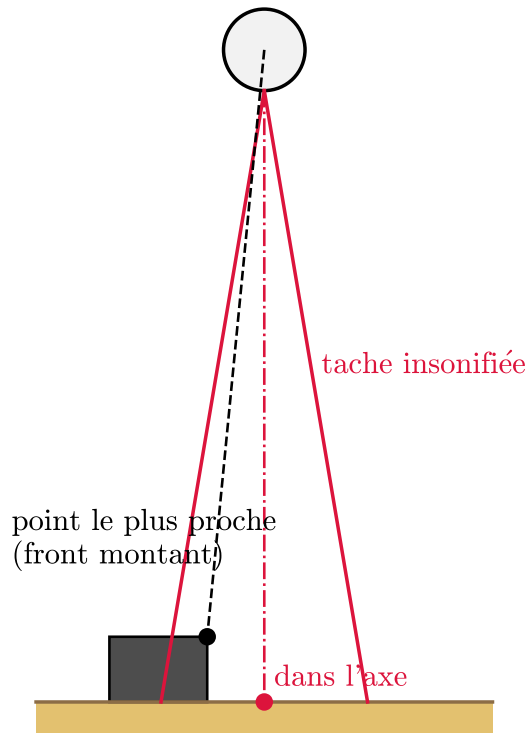


FIGURE 2.6 – Au bord d’un objet, le point le plus proche de la tache n’est pas dans l’axe du faisceau (échelle dilatée)

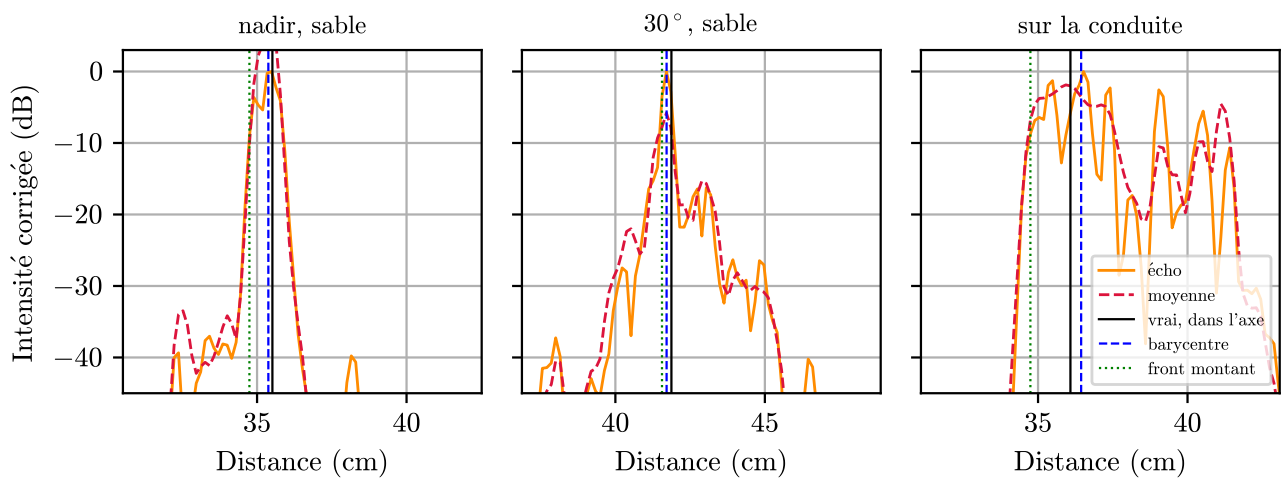


FIGURE 2.7 – Échos simulés à la verticale d’un fond de sable, de biais, et au bord d’une conduite. L’écho d’une impulsion (orange) fluctue autour de l’écho moyen (tirets rouges).

Chapitre 3

Orientation du faisceau par une sphère

Ce chapitre présente le dispositif qui oriente le faisceau. Le transducteur est fixé dans une sphère étanche, entièrement immergée, que trois roues font tourner sur elle-même. Nous décrivons d'abord le principe du dispositif, puis nous étudions l'équilibre de la sphère dans l'eau et son entraînement par les roues. Nous présentons ensuite la représentation de son orientation, la mesure de celle-ci par une centrale inertielle et son asservissement.

3.1 Principe du dispositif

Vue d'ensemble

La figure 3.1 montre le montage étudié, à l'échelle. Une sphère de 10 cm de diamètre, qui contient le transducteur, est immergée dans un bassin, 40 cm au-dessus d'un fond en relief. Elle est tenue par un berceau qui la fait tourner sur elle-même ; un câble relie le berceau à un ordinateur. Le faisceau sort de la sphère selon un rayon, dans la direction $\vec{b}$. L'écho revient au bout d'un temps qui donne la distance r entre la surface de la sphère et le fond : le point du fond qui a renvoyé l'écho est donc en $O + (R + r)\vec{b}$, où O est le centre de la sphère et R son rayon. En faisant tourner la sphère, on balaie le fond point par point.

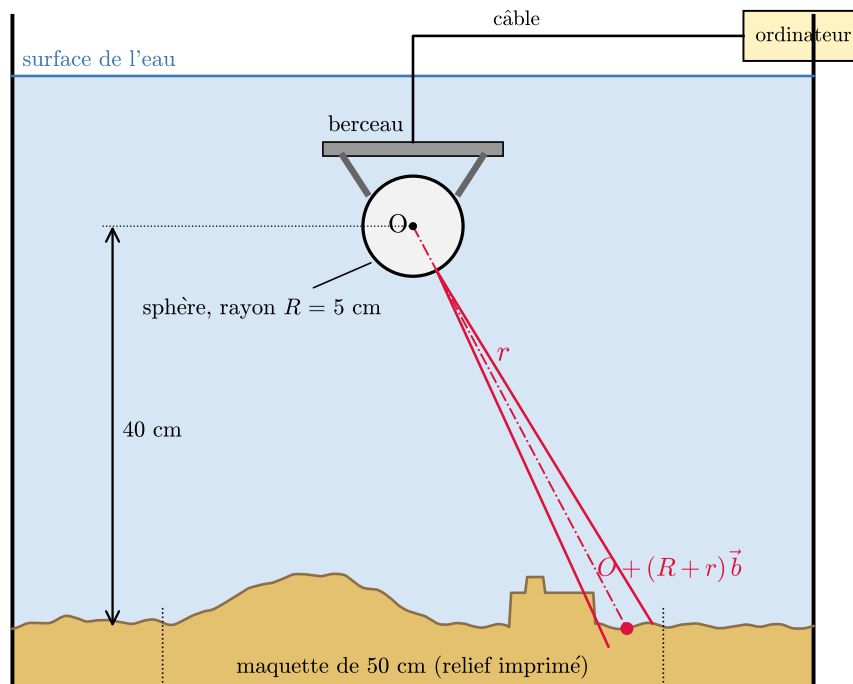


FIGURE 3.1 – Le montage dans le bassin, en coupe et à l'échelle : la sphère dans son berceau, son faisceau, et le fond en relief

Intérêt d'une sphère

La solution la plus courante pour orienter un capteur est une tourelle à deux axes : une première rotation

autour d'un axe vertical, puis une inclinaison. Elle présente deux inconvénients. D'une part, elle a une direction singulière, celle du premier axe : pour un axe vertical, c'est la verticale, que le sonar vise le plus souvent. Quand le faisceau passe près de la verticale, un petit déplacement demande une grande rotation du premier axe : pour balayer à 0,87 rad/s une ligne qui passe à 0,5° de la verticale, celle-ci doit tourner à 100 rad/s (Fig 3.2). D'autre part, les axes de rotation ne passent pas par le transducteur, qui se déplace quand on l'oriente.

Une sphère qui tourne sur elle-même n'a pas ces défauts. Le transducteur affleure à sa surface et rayonne selon le rayon : tous les faisceaux passent par le centre O de la sphère, qui reste fixe. Le point visé à la distance r est simplement $O + (R + r)\vec{b}$, où R est le rayon de la sphère et $\vec{b}$ la direction du faisceau. La sphère peut tourner autour de n'importe quel axe, avec la même cinématique dans toutes les orientations.

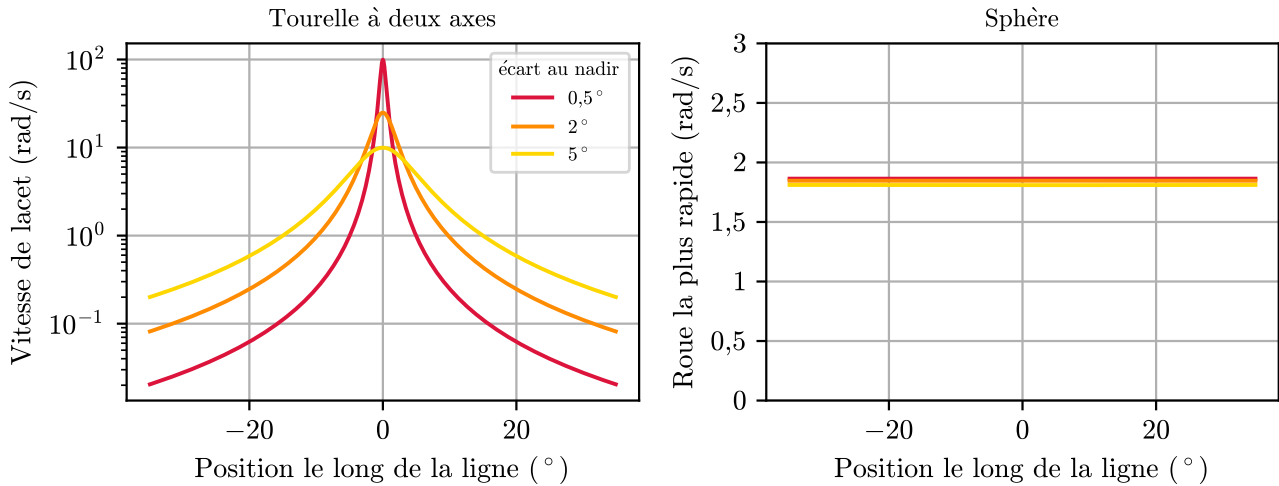


FIGURE 3.2 – Balayage d'une ligne à 0,87 rad/s passant à 0,5°, 2° et 5° de la verticale : vitesse du premier axe d'une tourelle à deux axes (à gauche) et vitesse de la roue la plus sollicitée de la sphère (à droite)

Description

La sphère est une boule creuse en acrylique transparent de 100 mm de diamètre ($R = 50$ mm) et de 150 g. Le disque piézoélectrique est collé au pôle sud, affleurant. La centrale inertielle (gyromètre et accéléromètre) est au centre. Une carte Raspberry Pi Pico, une batterie, le générateur d'impulsions et la chaîne de réception occupent le reste du volume. La batterie se recharge par induction : pour cela, la sphère tourne sa bobine vers celle du berceau.

La sphère est entièrement immergée dans un berceau (Fig 3.3). Trois roues omnidirectionnelles de 38 mm de diamètre appuient sur son hémisphère supérieure, à 120° les unes des autres et à l'angle α du pôle nord. Une roue omnidirectionnelle porte sur sa jante de petits galets libres, dont l'axe est tangent à la jante : elle entraîne comme une roue ordinaire dans le sens de sa rotation, mais n'oppose aucune résistance à un déplacement selon son axe, que les galets absorbent en roulant (Fig 3.5). Chaque roue pousse ainsi la sphère selon la tangente horizontale au point de contact, sans gêner les mouvements que lui imposent les deux autres. Chaque moteur pas à pas est dans un boîtier étanche et entraîne sa roue à travers la paroi par un accouplement magnétique, sans joint tournant. La sphère communique avec le berceau par une liaison optique à travers sa paroi transparente : des diodes électroluminescentes et des photodiodes réparties autour de l'électronique, de sorte qu'au moins une paire voie la tête optique du berceau dans toute orientation. Seul le berceau est relié à la surface, par un câble.

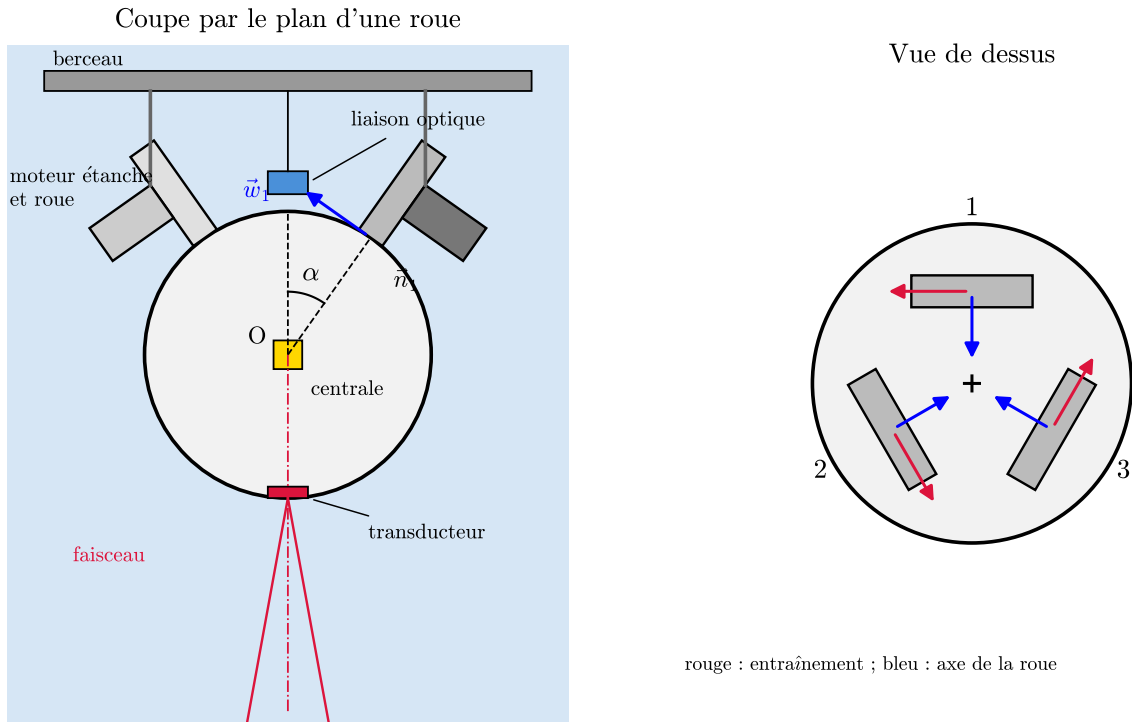


FIGURE 3.3 – La sphère dans son berceau : coupe par le plan d’une roue et vue de dessus. Chaque roue entraîne la sphère selon un parallèle (rouge) et laisse glisser ses galets selon le méridien ; $\vec{w}_i$ est l’axe de la roue i .

3.2 Équilibre de la sphère dans l’eau

Hypothèses

- La sphère est entièrement immergée et indéformable ;
- son centre O est fixe : elle ne fait que tourner sur elle-même ;
- son centre de gravité G est à la distance d de O , du côté du transducteur ;
- les roues appuient au-dessus de l’équateur, avec un coefficient de frottement μ .

Forces de pression

Les forces de pression sont normales à la surface de la sphère : elles passent toutes par son centre. Leur résultante, la poussée d’Archimède, n’exerce donc aucun couple autour de O , quelle que soit l’orientation de la sphère. De plus, une sphère qui tourne sur elle-même ne déplace pas d’eau : il n’y a pas de masse ajoutée. Seul le poids, appliqué en G , peut créer un couple.

Maintien contre les roues

La poussée vaut $\rho g \frac{4}{3} \pi R^3 = 5,13$ N, pour un poids de 1,47 N. La sphère est plus légère que l’eau qu’elle déplace : la différence, 3,66 N, la plaque contre les trois roues placées au-dessus d’elle. Si $\vec{n}_i$ est la normale au contact i , inclinée de α sur la verticale, l’équilibre vertical donne l’effort normal sur chaque roue :

$$N = \frac{\rho g V - mg}{3 \cos \alpha} \quad (3.1)$$

soit $N = 1,49$ N. Poussée vers le haut contre trois appuis, la sphère est aussi tenue latéralement. Cet effort ne dépend ni de la profondeur ni du niveau de l’eau ; on le règle par la masse de la sphère, qui doit rester inférieure à 523 g (Fig 3.4). Une sphère plus lourde que l’eau conviendrait aussi, avec les roues placées en dessous.

Frottement aux contacts

Chaque roue transmet un effort tangentiel au plus égal à μN . Comme nous le verrons, cet effort crée un couple selon l’axe de la roue, au plus égal à $\mu N R$. Avec $\mu \approx 0,4$ pour un galet en caoutchouc sur l’acrylique mouillé, on obtient 30 mN.m. Le moment d’inertie de la sphère étant de l’ordre de $1,6 \times 10^{-4}$ kg.m², elle peut accélérer jusqu’à 187 rad/s² sans que les roues glissent.

Couple dû au poids

Si le centre de gravité est à la distance d sous le centre, du côté du transducteur, le poids exerce un couple $mgd \sin \tau$ quand le faisceau est incliné de τ . Ce couple ramène le faisceau vers la verticale : si les moteurs

sont libérés, la sphère se redresse d'elle-même, transducteur vers le fond. Il consomme cependant une part du frottement disponible : avec $d = 2$ mm, il vaut 2,08 mN.m à 45°, soit 7% du couple transmissible.

Frottement visqueux

Quand la sphère tourne à la vitesse Ω , l'eau qui la touche est entraînée dans une couche limite d'épaisseur $\delta \sim \sqrt{\nu/\Omega}$, de l'ordre du millimètre. La contrainte $\mu_e R \Omega / \delta$ sur toute la surface donne un couple de l'ordre de $\frac{8\pi}{3} \mu_e R^4 \Omega / \delta$, soit 52 $\mu\text{N.m}$ à 1 rad/s, environ 600 fois moins que le couple que transmettent les roues. On le néglige.

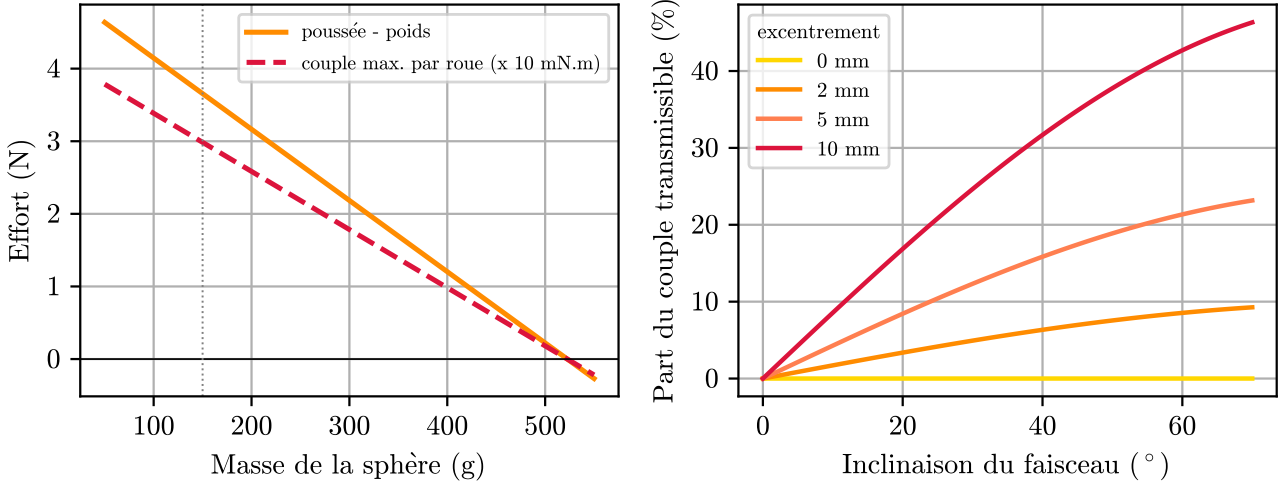


FIGURE 3.4 – Effort qui plaque la sphère contre les roues et couple transmissible en fonction de la masse de la sphère (à gauche), et part du couple transmissible consommée par le poids en fonction de l'excentrement du centre de gravité (à droite)

3.3 Entraînement par trois roues

Condition de roulement sans glissement

La sphère tourne à la vitesse angulaire $\vec{\omega}$. Au point de contact $R\vec{n}_i$, sa surface se déplace à la vitesse $\vec{\omega} \wedge R\vec{n}_i$. La roue i , de rayon r , tourne à la vitesse $\dot{\varphi}_i$ autour de son axe $\vec{w}_i = \vec{n}_i \wedge \vec{t}_i$, où $\vec{t}_i$ est sa direction d'entraînement ; sa jante se déplace à la vitesse $-r\dot{\varphi}_i \vec{t}_i$. Le roulement sans glissement selon $\vec{t}_i$ s'écrit :

$$-r\dot{\varphi}_i = (\vec{\omega} \wedge R\vec{n}_i) \cdot \vec{t}_i = R\vec{\omega} \cdot (\vec{n}_i \wedge \vec{t}_i) = R\vec{\omega} \cdot \vec{w}_i \quad (3.2)$$

La composante de la vitesse selon l'autre direction tangente, $\vec{w}_i$, est absorbée par les galets (Fig 3.5). Chaque roue impose donc la composante de la rotation de la sphère selon son propre axe.

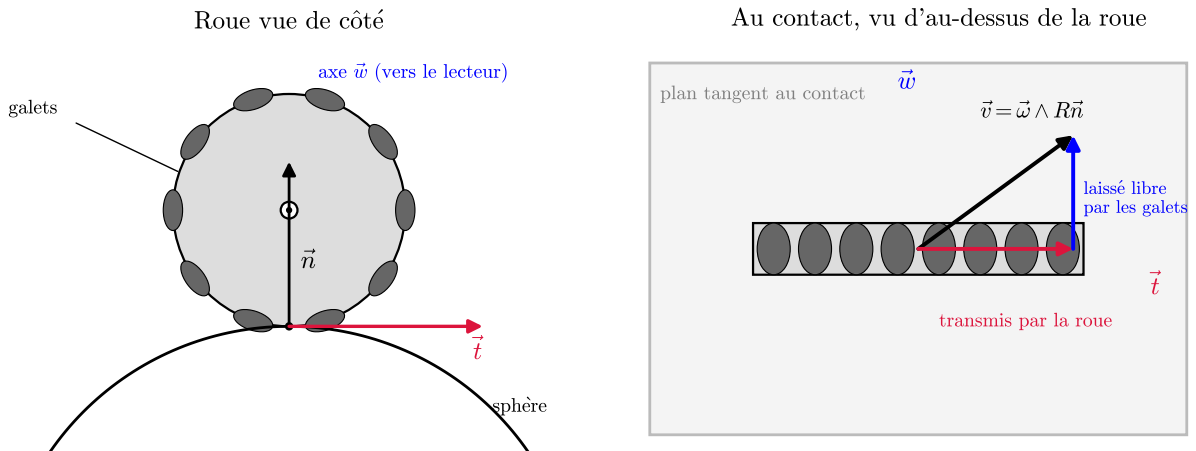


FIGURE 3.5 – Une roue omnidirectionnelle sur la sphère. À gauche, la roue vue de côté : $\vec{n}$ est la normale au contact, $\vec{t}$ la direction d'entraînement, $\vec{w}$ l'axe de la roue. À droite, le plan tangent au contact : de la vitesse $\vec{v}$ de la surface de la sphère, la roue ne transmet que la composante selon $\vec{t}$.

De même, l'effort tangentiel F_i de la roue i crée le couple $R\vec{n}_i \wedge F_i\vec{t}_i = RF_i\vec{w}_i$, selon son axe.

Relation entre les vitesses des roues et la rotation de la sphère

En notant W la matrice dont les lignes sont les $\vec{w}_i$, on obtient :

$$\dot{\vec{\varphi}} = -\frac{R}{r} W \vec{\omega} \quad \Longleftrightarrow \quad \vec{\omega} = -\frac{r}{R} W^{-1} \dot{\vec{\varphi}} \quad (3.3)$$

Trois roues suffisent pour imposer n'importe quelle rotation, pourvu que leurs axes soient indépendants.

Choix de l'angle des contacts

Avec des contacts à l'angle α du pôle et à 120° les uns des autres, l'axe de la roue i est tangent au méridien : $\vec{w}_i = (-\cos \alpha \cos \psi_i, -\cos \alpha \sin \psi_i, \sin \alpha)$. Le produit scalaire de deux axes vaut :

$$\vec{w}_i \cdot \vec{w}_j = \cos^2 \alpha \cos 120^\circ + \sin^2 \alpha = \sin^2 \alpha - \frac{1}{2} \cos^2 \alpha \quad (3.4)$$

Il s'annule pour $\tan \alpha = 1/\sqrt{2}$, soit $\alpha = 35,26^\circ$. Les trois axes sont alors orthogonaux, comme les trois arêtes d'un coin de cube, et W est une matrice de rotation : $W^{-1} = W^T$. Chaque roue commande une composante de la rotation indépendamment des autres et toutes les directions de rotation demandent la même vitesse aux roues. On mesure cette propriété par le conditionnement de W , rapport entre la plus grande et la plus petite vitesse des roues nécessaires pour faire tourner la sphère à une vitesse donnée, selon la direction de l'axe de rotation : il vaut 1 à cet angle et augmente dès que l'on s'en écarte (Fig 3.6). Un conditionnement élevé signifierait que certaines rotations demandent des vitesses de roues bien plus grandes que d'autres, et que les erreurs des roues y sont amplifiées. Nous plaçons donc les contacts à cet angle.

Résolution angulaire

Avec des roues de 19 mm de rayon, $R/r = 2,63$. Un moteur de 200 pas par tour commandé en seizièmes de pas fait 3200 micropas par tour : un micropas tourne la sphère de $0,043^\circ$, ce qui est largement suffisant.

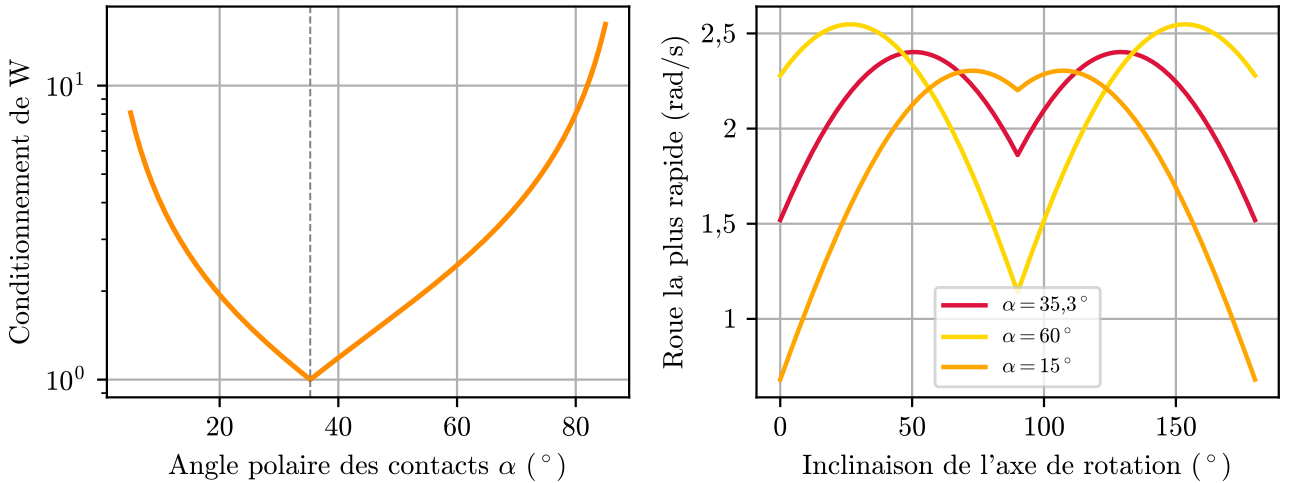


FIGURE 3.6 – Conditionnement de la matrice W en fonction de l'angle des contacts (à gauche) et vitesse de la roue la plus sollicitée pour une rotation de 1 rad/s, en fonction de l'inclinaison de l'axe de rotation (à droite)

3.4 Orientation de la sphère et pointage du faisceau

Pour viser une direction donnée, il faut faire tourner la sphère d'une certaine rotation. Cette section explique comment on décrit l'orientation de la sphère, comment on choisit la rotation qui pointe le faisceau vers une direction voulue, et comment on enchaîne les directions pour balayer le fond (Fig 3.7).

Deux repères

On utilise deux repères de même origine O , le centre de la sphère : le repère du bassin, fixe, dont l'axe $\vec{e}_z$ est vertical vers le haut, et le repère de la sphère, qui tourne avec elle. Dans le repère de la sphère, le transducteur rayonne toujours selon le même vecteur, $\vec{e}_b = -\vec{e}_z$: vers le bas quand la sphère est au repos. L'orientation de la sphère est la rotation qui fait passer de son repère à celui du bassin ; la direction du faisceau, dans le bassin, est l'image de $\vec{e}_b$ par cette rotation.

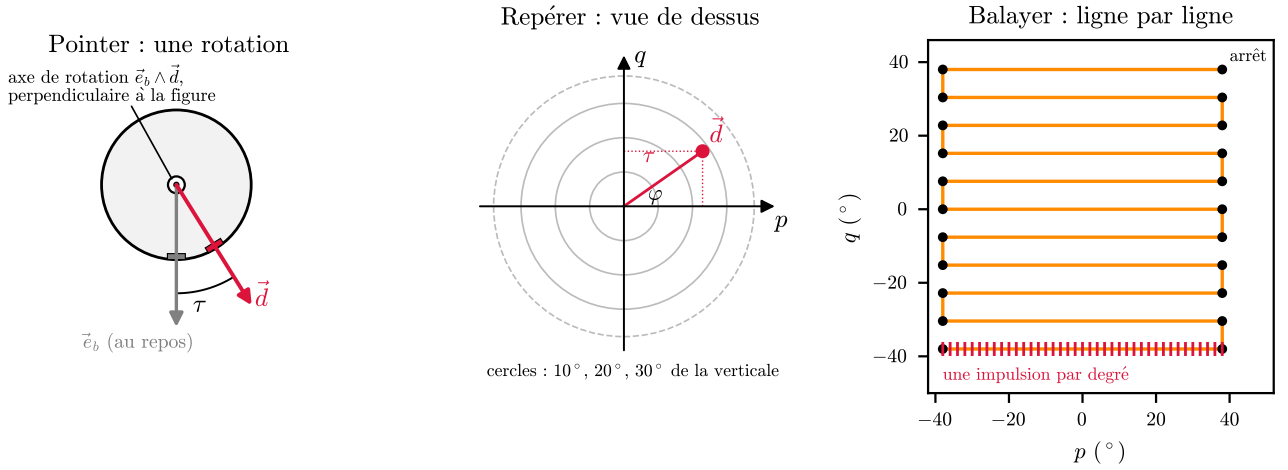


FIGURE 3.7 – À gauche : pour pointer le faisceau vers $\vec{d}$, la sphère tourne de l'angle τ autour d'un axe horizontal. Au centre : une direction est repérée, vue de dessus, par son angle τ à la verticale et son azimut φ , ou par les coordonnées (p, q) . À droite : le balayage parcourt les directions ligne par ligne (une ligne sur huit est dessinée).

Représentation de l'orientation

Toute rotation peut se décrire par un axe $\vec{u}$ et un angle ϕ . On la représente par un quaternion unitaire, quatre nombres qui regroupent cet axe et cet angle [16] :

$$q = \left(\cos \frac{\phi}{2}, \sin \frac{\phi}{2} \vec{u} \right) \quad (3.5)$$

Les quaternions se multiplient selon une règle qui traduit l'enchaînement de deux rotations. Un vecteur $\vec{v}$ du repère de la sphère devient $q\vec{v}q^*$ dans le repère du bassin, où q^* est le quaternion de la rotation inverse ; en particulier, le faisceau pointe dans la direction $\vec{b} = q\vec{e}_b q^*$. Si $\vec{\omega}$ est la vitesse angulaire de la sphère, l'orientation évolue selon $\dot{q} = \frac{1}{2}\vec{\omega}q$. Contrairement à une description par trois angles successifs, les quaternions n'ont pas de direction singulière.

Rotation qui pointe le faisceau

Pour pointer vers une direction $\vec{d}$, une infinité d'orientations conviennent : une fois le faisceau dans la direction $\vec{d}$, on peut encore faire tourner la sphère autour de $\vec{d}$ sans le déplacer. Le faisceau étant symétrique autour de son axe, cette dernière rotation est sans effet. On choisit la plus simple, celle qui amène $\vec{e}_b$ sur $\vec{d}$ par une seule rotation d'angle τ , l'angle entre la verticale et $\vec{d}$, autour de l'axe horizontal $\vec{e}_b \wedge \vec{d}$ (Fig 3.7, à gauche). Son quaternion s'écrit :

$$q(\vec{d}) = \frac{(1 + \vec{e}_b \cdot \vec{d}, \vec{e}_b \wedge \vec{d})}{\|(1 + \vec{e}_b \cdot \vec{d}, \vec{e}_b \wedge \vec{d})\|} \quad (3.6)$$

Elle est définie pour toute direction, sauf vers le haut, qui ne sert pas.

Repérage des directions

Vue de dessus, une direction du faisceau est repérée par son angle τ à la verticale et par son azimut φ (Fig 3.7, au centre). Pour balayer un carré, on utilise plutôt les coordonnées $p = \tau \cos \varphi$ et $q = \tau \sin \varphi$: la verticale correspond à $(0, 0)$, et deux directions voisines sont séparées d'un angle égal, à quelques pour cent près, à leur distance en (p, q) .

Trajectoire de balayage

Le balayage parcourt un carré de $\pm 38^\circ$ en (p, q) , ligne par ligne, dans un sens puis dans l'autre (Fig 3.7, à droite). Les lignes sont parcourues à 0,87 rad/s et espacées de 1° . La sonde émet 50 impulsions par seconde, soit une tous les degrés le long d'une ligne. À chaque coin, la sphère s'arrête 0,15 s ; nous verrons que ces arrêts servent à la mesure de l'orientation. Le balayage complet dure 166 s et compte 8333 impulsions.

3.5 Mesure de l'orientation

Pour placer chaque mesure au bon endroit du fond, il faut connaître la direction du faisceau à l'instant de chaque impulsion, au dixième de degré près. Les moteurs ne suffisent pas : les roues peuvent glisser légèrement sur la sphère. On mesure donc l'orientation directement dans la sphère, avec une centrale inertielle.

Centrale inertielle

La centrale inertielle est un petit composant qui contient un gyromètre et un accéléromètre, chacun sur trois axes. Elle est fixée au centre de la sphère et tourne avec le transducteur.

- Le gyromètre mesure la vitesse angulaire de la sphère. En l'intégrant, on obtient l'orientation, mais les petites erreurs de mesure s'accumulent : l'orientation calculée dérive peu à peu. L'erreur principale est un biais, une petite vitesse mesurée même à l'arrêt.
- L'accéléromètre mesure l'accélération, plus la pesanteur. Comme le centre de la sphère ne bouge pas, il ne mesure que la pesanteur, quelle que soit la rotation : il donne à tout instant la verticale dans le repère de la sphère, donc l'inclinaison. Mais il ne dit rien d'une rotation autour de la verticale, qui laisse la pesanteur inchangée.

Filtre complémentaire

Les deux capteurs se complètent : le gyromètre est précis sur de courtes durées mais dérive, l'accéléromètre ne dérive pas mais ne voit que l'inclinaison. Le filtre complémentaire [17] intègre le gyromètre et corrige lentement l'orientation obtenue à l'aide de l'accéléromètre (Fig 3.8). Si $\vec{a}$ est la verticale mesurée par l'accéléromètre et $\hat{v} = \hat{q}^* \vec{e}_z \hat{q}$ la verticale prédite par l'orientation estimée $\hat{q}$, toutes deux dans le repère de la sphère, l'estimation évolue selon :

$$\dot{\hat{q}} = \frac{1}{2} \hat{q} \left(\vec{\omega}_{gyro} - \hat{\beta} + k_p \vec{a} \wedge \hat{v} \right), \quad \dot{\hat{\beta}} = -k_i \vec{a} \wedge \hat{v} \quad (3.7)$$

où $\hat{\beta}$ est le biais estimé du gyromètre. Le produit vectoriel $\vec{a} \wedge \hat{v}$ est un vecteur rotation : il fait tourner la verticale prédite vers la verticale mesurée, d'autant plus vite que l'écart est grand. Les gains k_p et k_i fixent la rapidité de ces corrections.

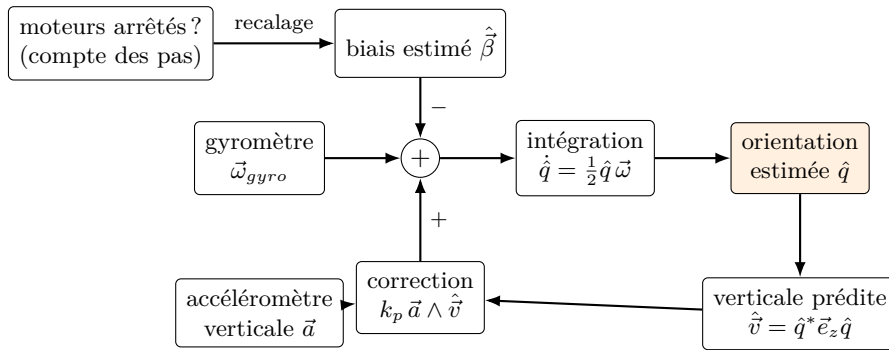


FIGURE 3.8 – Le filtre complémentaire : le gyromètre est intégré, l'accéléromètre corrige l'inclinaison, et le biais du gyromètre est recalé quand les moteurs sont arrêtés

Correction du biais à l'arrêt

L'accéléromètre ne corrigeant pas la rotation autour de la verticale, le biais du gyromètre selon cet axe ferait dériver l'orientation sans limite. On utilise pour cela les moteurs : lorsque le compte des pas indique que les trois moteurs sont arrêtés, la sphère est immobile et ce que mesure le gyromètre n'est que son biais. On le recalé alors sur les trois axes ; c'est la « mise à jour à vitesse nulle » utilisée en navigation inertielle [18]. Les arrêts aux coins du balayage, toutes les deux secondes environ, y suffisent. L'orientation initiale autour de la verticale n'est mesurée par rien : elle fixe l'orientation de la carte et se repère par une marque sur la sphère.

Apport de chaque correction

La figure 3.9 compare, sur le balayage simulé complet, trois versions du filtre ; on donne l'erreur sur la direction du faisceau que dépassent 5% des impulsions. Avec le gyromètre seul, l'orientation dérive dans toutes les directions et l'erreur atteint 2,4°. L'accéléromètre corrige l'inclinaison et ramène l'erreur à 0,6°, mais la rotation autour de la verticale continue de dériver : 0,6° en fin de balayage. Avec le recalage du biais à chaque arrêt, cette dérive n'est plus que de 0,13° et l'erreur tombe à 0,35° ; ce qui reste vient surtout de l'erreur de 0,3° sur l'orientation initiale autour de la verticale, que la simulation introduit volontairement.

Étalonnage de l'accéléromètre

Un biais de l'accéléromètre fausse directement l'inclinaison : 1 mg de biais incline la verticale de 0,06°. On l'étalonne en mesurant la pesanteur au repos dans de nombreuses orientations, que la sphère prend d'elle-même : le biais et le facteur d'échelle de chaque axe sont ceux qui donnent la même norme g dans toutes les orientations.

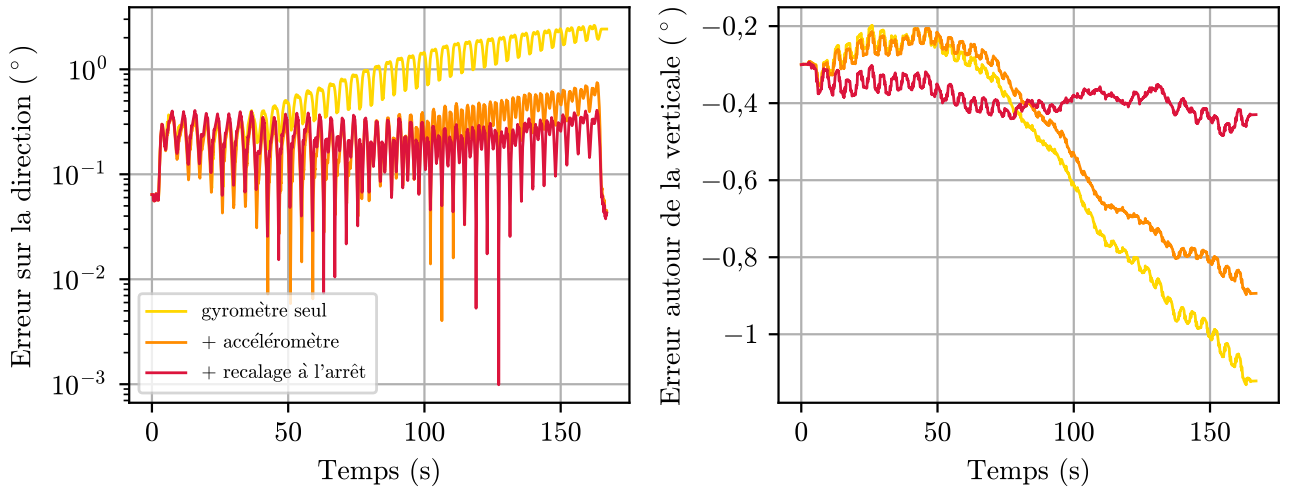


FIGURE 3.9 – Erreur sur la direction du faisceau (à gauche) et sur la rotation autour de la verticale (à droite) pendant le balayage simulé, pour trois versions du filtre

3.6 Asservissement de l'orientation

L'asservissement fait suivre à la sphère l'orientation voulue à chaque instant, malgré le glissement des roues et les erreurs des moteurs. Il compare l'orientation voulue à l'orientation estimée et en déduit la vitesse angulaire à donner à la sphère (Fig 3.10).

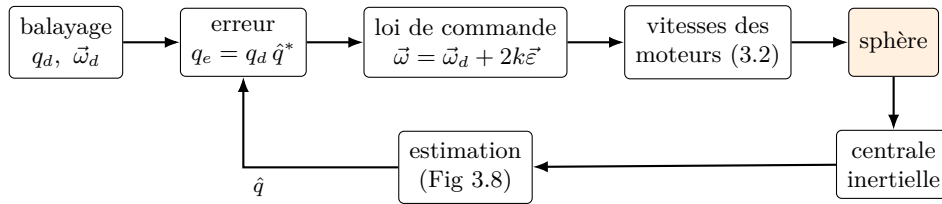


FIGURE 3.10 – La boucle d'asservissement de l'orientation, calculée cent fois par seconde par l'ordinateur

Loi de commande

On note q_d l'orientation voulue, $\vec{\omega}_d$ sa vitesse de variation, connue d'après la trajectoire de balayage, et $q_e = q_d \hat{q}^* = (\eta, \vec{\varepsilon})$ l'erreur, c'est-à-dire la rotation qui reste à faire, choisie avec $\eta \geq 0$. Pour une petite erreur d'angle θ_e , $2\vec{\varepsilon}$ est le vecteur rotation correspondant, de norme θ_e . On commande :

$$\vec{\omega} = \vec{\omega}_d + 2k\vec{\varepsilon} \quad (3.8)$$

La sphère suit la trajectoire grâce au premier terme et rattrape son retard grâce au second. On en déduit les vitesses des moteurs par la relation (3.2). La vitesse est limitée à 1,5 rad/s et l'accélération à 10 rad/s², bien en dessous de ce que transmet le frottement.

Stabilité

Avec $\dot{q} = \frac{1}{2}\vec{\omega}q$ et $\dot{q}_d = \frac{1}{2}\vec{\omega}_d q_d$, l'erreur vérifie $\dot{q}_e = \frac{1}{2}(\vec{\omega}_d q_e - q_e \vec{\omega})$, dont la partie scalaire donne $\dot{\eta} = k\|\vec{\varepsilon}\|^2$. La fonction $V = 1 - \eta$, positive et nulle seulement quand l'orientation voulue est atteinte, vérifie donc $\dot{V} = -k\|\vec{\varepsilon}\|^2 \leq 0$: l'erreur tend vers zéro [19]. Près de l'orientation voulue, l'angle d'erreur décroît comme e^{-kt} ; nous prenons $k = 5 \text{ s}^{-1}$, soit un temps de réponse de 0,2 s.

Retards et horloges

Les mesures de la centrale arrivent à l'ordinateur avec quelques millisecondes de retard, par la liaison optique puis par le câble du berceau ; le compte des pas arrive avec 1 ms de retard. Pour savoir si une mesure a été prise à l'arrêt, il faut attendre que le compte des pas correspondant soit arrivé : l'estimation traite donc les mesures avec 40 ms de retard, et la commande prolonge la dernière estimation jusqu'à l'instant présent avec les vitesses qu'elle a commandées. Chaque carte date ses messages avec sa propre horloge ; l'ordinateur en déduit le décalage de chaque horloge par rapport à la sienne en retenant le message arrivé avec le moins de retard.

Résultats en simulation

La figure 3.11 montre un balayage complet, piloté par le programme de commande à travers le simulateur décrit au chapitre suivant. Le simulateur reproduit le bruit et le biais du gyromètre et de l'accéléromètre, un

glissement de 1% au contact des roues, des retards de 2 à 8 ms, des pertes de messages et des horloges décalées. La direction estimée du faisceau s'écarte de la direction vraie de $0,22^\circ$ en moyenne et de moins de $0,35^\circ$ pour 95% des impulsions, soit moins du dixième de l'ouverture du faisceau. La simulation part d'une orientation initiale fautive de $0,3^\circ$ autour de la verticale, que rien ne mesure : c'est l'essentiel de l'erreur, qui fait tourner la carte d'un bloc. Le biais du gyromètre est retrouvé à chaque arrêt et les roues n'utilisent jamais plus de 22% du frottement disponible.

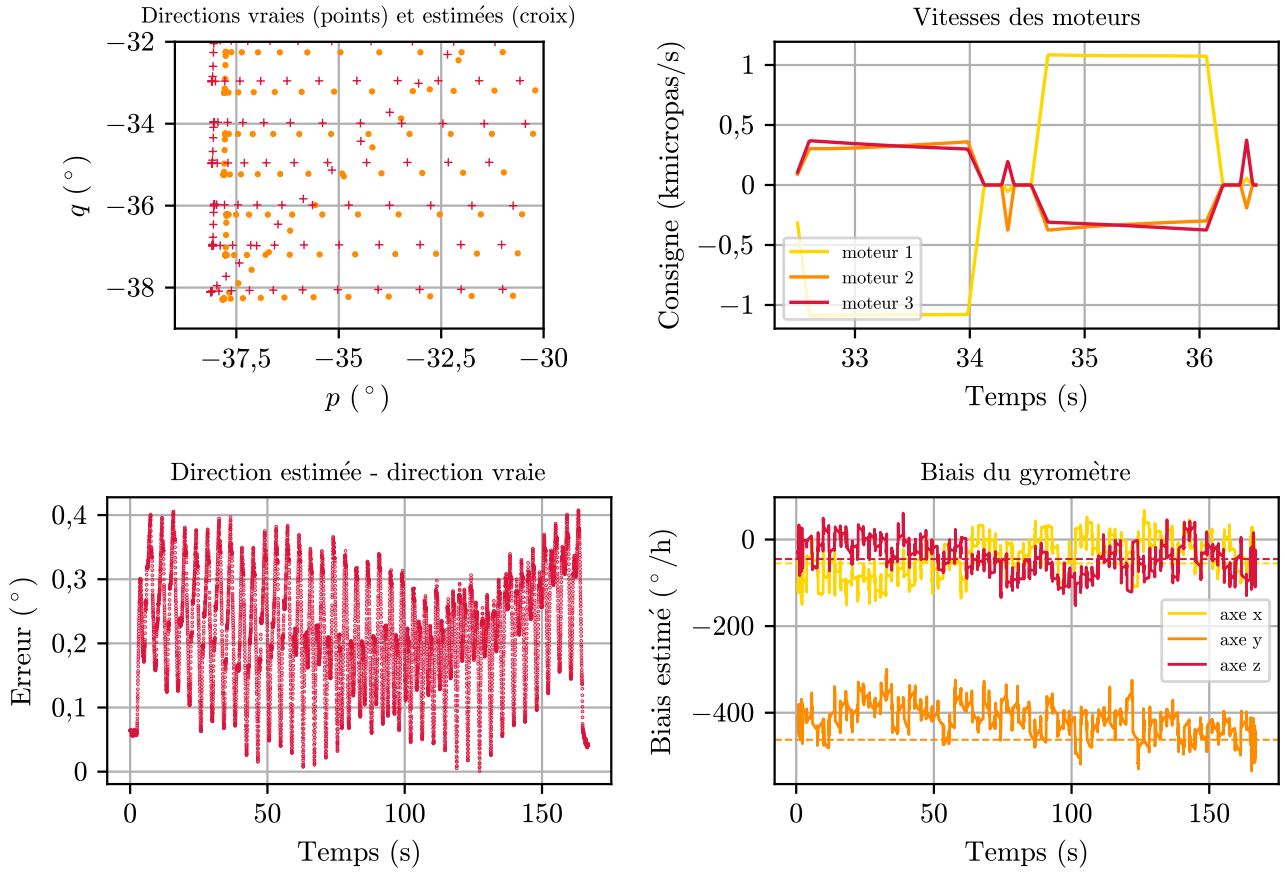


FIGURE 3.11 – Balayage simulé : directions des impulsions, consignes des moteurs pendant une ligne et un changement de ligne, écart entre direction estimée et direction vraie, biais du gyromètre estimé (traits pleins) et vrai (tirets)

Chapitre 4

Simulation de l'acquisition et imagerie du fond

Ce chapitre présente les programmes de simulation et leurs résultats. Le principe est le suivant : on se donne un fond connu, on simule les données que fournirait le dispositif en le balayant, puis on reconstruit le fond à partir de ces seules données, comme on le ferait avec le dispositif réel. La comparaison entre le fond reconstruit et le fond connu mesure la fidélité de l'image. Dans un premier temps, nous créons un relief et nous modélisons l'environnement. Nous simulons ensuite l'acquisition, puis nous reconstruisons le fond par une méthode classique et par une méthode de déconvolution. Enfin, nous caractérisons la nature du fond à partir de l'énergie des échos. Les programmes sont écrits en Python (NumPy, SciPy, Matplotlib).

4.1 Organisation des programmes

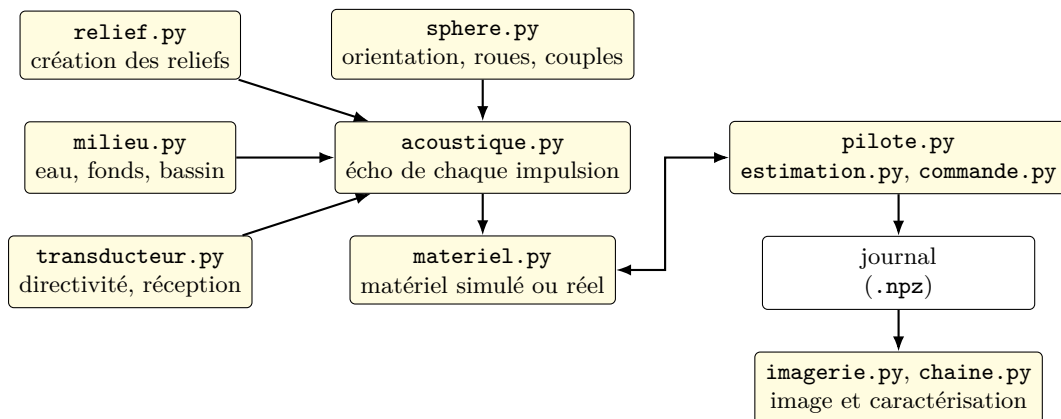


FIGURE 4.1 – Organisation des programmes

Les modules du dossier `programmes/sonar` s'organisent comme sur la figure 4.1. Le programme qui pilote le balayage, `pilote.py`, est le même pour la simulation et pour le système réel : il envoie des consignes de vitesse aux moteurs et reçoit des messages datés (odométrie, mesures de la centrale, impulsions). Le module `materiel.py` fournit deux matériels qui répondent de la même façon : le simulateur et les cartes électroniques réelles. Les programmes d'imagerie ne lisent que le journal d'acquisition ; ils traitent donc de la même façon des données simulées et réelles.

4.2 Création des reliefs

Principe

Un relief est un champ de hauteurs $z(x, y)$ défini sur une grille de 2 mm, avec pour chaque maille un type de fond. La maquette de référence mesure 50×50 cm ; elle est placée au centre d'un fond de 80×80 cm, 40 cm sous le centre de la sphère (Fig 4.2). Elle réunit ce que l'on cherche sur un fond marin :

- un relief naturel : un mouvement brownien fractionnaire obtenu par synthèse spectrale [20], d'exposant de Hurst 0,8 et d'écart type 4 mm, une butte de 5 cm et une cuvette de 2 cm ;
- des rides de sable de 4 cm de longueur d'onde et 2 mm d'amplitude, de l'ordre de la taille de la tache ;
- un chenal de 5 cm de large et 2,5 cm de profondeur, envasé ;
- cinq rochers de 1,5 à 3,5 cm de haut ;
- une conduite de 4 cm de diamètre, enterrée de 8 mm ;
- une épave de 16 × 5 cm et 3,5 cm de haut, aux flancs verticaux ;
- quatre types de fond : du sable, du gravier, de la vase dans la cuvette et le chenal, et de la roche pour les rochers, la conduite et l'épave.

Séparation du relief et de la rugosité

Le relief simulé ne contient pas de détails de moins d'un centimètre. La rugosité plus fine est déjà prise en compte dans le modèle de rétrodiffusion, par la pente δ et le paramètre de Lambert μ : la placer aussi dans le relief la compterait deux fois et aplatiserait la réponse angulaire des fonds.

Maquette réelle

Le programme `creer_relief.py` écrit aussi le relief en fichiers STL, découpé en tuiles imprimables. Le fond de la maquette réelle serait ainsi exactement celui de la simulation, ce qui permettrait de comparer directement les mesures et les simulations.

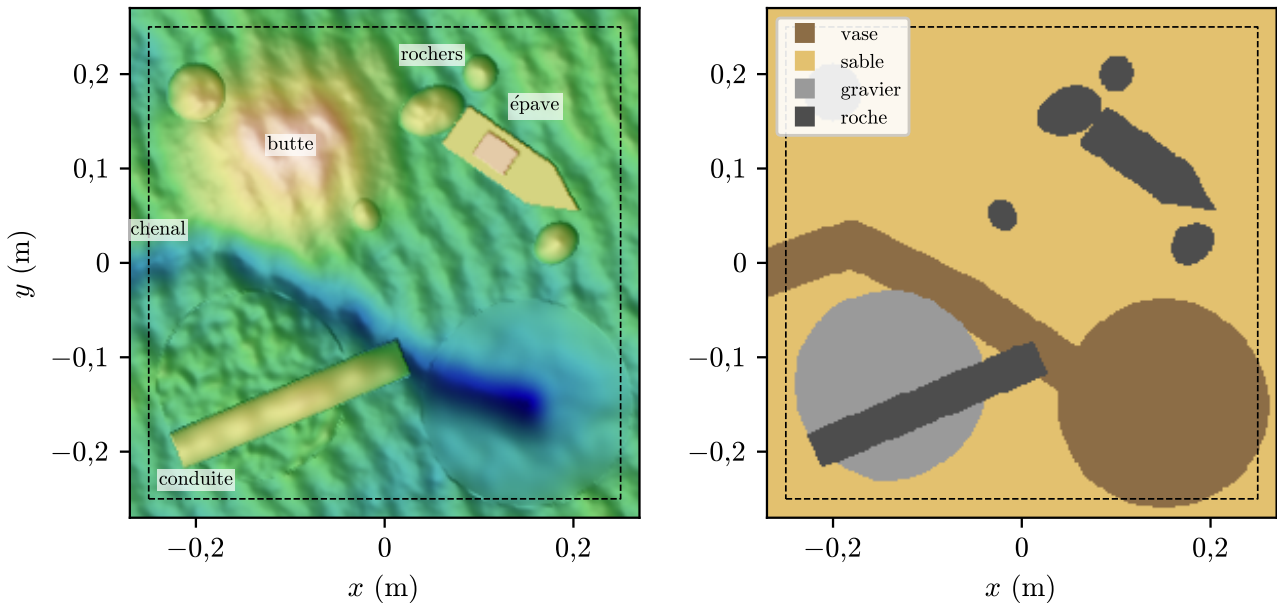


FIGURE 4.2 – La maquette de référence : relief (à gauche) et types de fond (à droite). Le carré en tirets délimite la maquette de 50 × 50 cm.

4.3 Modélisation de l'environnement

Le module `milieu.py` décrit l'eau, dont la célérité (2.2) et l'absorption dépendent de la température et de la salinité, les quatre types de fond et leur rétrodiffusion (Fig 2.5), et le bassin. Avant toute impulsion, le simulateur détermine les mailles vues depuis le centre de la sphère : il parcourt le segment qui relie chaque maille au centre et vérifie que le relief reste en dessous. Les rochers, l'épave et la conduite portent ainsi des ombres, qui ne renvoient aucun écho.

4.4 Simulation de l'acquisition

Calcul de l'écho

Pour une orientation q de la sphère, le faisceau part de la face du transducteur, en $O + R\vec{b}$. Chaque maille visible j située dans le cône du faisceau renvoie une onde dont l'amplitude suit l'équation du sonar :

$$p_j = p_0 D(\psi_j)^2 \frac{\sqrt{\sigma(\theta_j) A_j}}{r_j^2} 10^{-\alpha r_j/10} \xi_j e^{-2ikr_j} \quad (4.1)$$

où ψ_j est l'angle à l'axe, θ_j l'incidence locale, A_j l'aire vraie de la maille et ξ_j un coefficient complexe gaussien propre à la maille, tiré une fois pour toutes. Le signal reçu est la somme de ces ondes, chacune retardée de $2r_j/c$ et mise en forme par l'enveloppe de l'impulsion. Le coefficient ξ_j et la phase $2kr_j$ reproduisent le chatoiement. On ajoute le bruit du préamplificateur ; le détecteur logarithmique et le convertisseur analogique-numérique de 12 bits échantillonnent ensuite l'enveloppe à 500 kHz, soit 576 échantillons jusqu'à 85 cm. Une impulsion se calcule en 3 ms environ.

Simulation du matériel

Le simulateur fait tourner la sphère selon les vitesses réelles des moteurs, avec la rampe d'accélération de leur carte, un glissement de 1% au contact de chaque roue et un chien de garde ; il vérifie à chaque pas de calcul que le frottement suffit. Il produit les mesures de la centrale inertielle (bruit, biais, dérive du biais, quantification sur 16 bits), l'odométrie et les impulsions, avec des retards, des pertes de messages et des horloges décalées. C'est avec ce simulateur que le programme de pilotage a produit le balayage de la figure 3.11.

4.5 Reconstruction du relief

4.5.1 Méthode classique : détection du fond

Pour chaque impulsion, on corrige l'intensité de la divergence et de l'absorption, $I = v^2 r^4 10^{\alpha r/5}$, on cherche le maximum de l'écho et on prend le barycentre de l'énergie des échantillons contigus situés à moins de 6 dB de ce maximum. Le point du fond est placé dans l'axe du faisceau, en $O + (R + r)\vec{b}$, avec l'orientation estimée à l'instant de l'impulsion. On obtient un nuage de points, que l'on interpole sur la grille de 2 mm : c'est le modèle numérique de terrain (MNT), c'est-à-dire la carte des hauteurs du fond.

4.5.2 Méthode proposée : déconvolution des données

Idée

La méthode classique attribue tout l'écho d'une impulsion au seul point situé dans l'axe du faisceau. Or le faisceau a une largeur : l'écho mélange les contributions de tous les points de la tache, chacun pondéré par la directivité. Comme les impulsions sont émises tous les degrés et que le faisceau en mesure environ cinq, les taches voisines se recouvrent largement : chaque point du fond est vu par une vingtaine d'impulsions, chacune avec un poids différent (Fig 4.3, à gauche). C'est exactement la situation d'une photographie floue : chaque pixel de l'image mélange la lumière de ses voisins, selon une loi connue. Si l'on connaît cette loi, on peut en grande partie défaire le mélange : c'est la déconvolution. Ici, la loi du mélange est connue : c'est la directivité du transducteur, que l'on a calculée au chapitre 2 et que l'on mesurerait sur le dispositif réel.

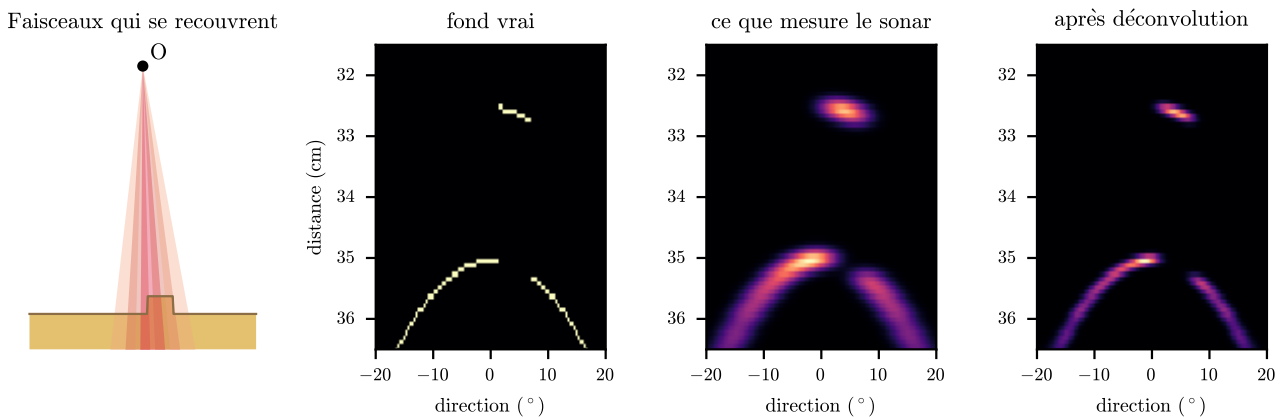


FIGURE 4.3 – Principe de la déconvolution, sur un exemple à une seule ligne de directions : un fond plat portant un objet. De gauche à droite : les faisceaux voisins se recouvrent ; le fond vu depuis le centre de la sphère (distance du fond dans chaque direction) ; ce que mesure le sonar, flou en direction et en distance ; le fond retrouvé après 20 itérations.

Modèle de la mesure

Tous les faisceaux partent du centre de la sphère : on peut ranger les données dans un tableau à trois dimensions, deux pour la direction du faisceau et une pour la distance. On note $X(\vec{u}, r)$ la réflectivité du fond dans la direction $\vec{u}$ à la distance r : pour chaque direction, elle n'est non nulle qu'à la distance du fond dans cette

direction. L'intensité moyenne reçue par l'impulsion i , de direction $\vec{b}_i$, vaut alors :

$$Y_i(r) = \int D^4(\vec{b}_i, \vec{u}) (P * X(\vec{u}, \cdot)) (r) d\Omega_u \quad (4.2)$$

où D^4 est la directivité aller-retour, en intensité, entre la direction du faisceau et la direction $\vec{u}$, et P l'intensité de l'impulsion en fonction de la distance : le fond est rendu flou en direction par le faisceau et en distance par la durée de l'impulsion. C'est un modèle linéaire, $Y = HX$, où la matrice H est connue. Le diagramme s'évalue depuis la face du transducteur, à R du centre : un point vu sous l'angle ψ depuis le centre est vu depuis la face sous un angle plus grand d'un facteur $(R + r)/r$, soit 14% à la verticale. On en tient compte à l'aide de la distance du fond donnée par la méthode classique.

Algorithme de Richardson-Lucy

Il s'agit de retrouver X connaissant Y et H . Une inversion directe amplifierait le bruit et donnerait des réflectivités négatives, sans signification. On utilise l'algorithme itératif de Richardson-Lucy [21, 22]. Il part d'une première estimation, la moyenne des données pondérée par le diagramme, et la corrige à chaque itération en comparant les données à ce que l'estimation prédit, tout en conservant une solution positive :

$$X \leftarrow X \cdot \frac{H^T (Y/HX)}{H^T \mathbf{1}} \quad (4.3)$$

Là où l'estimation prédit trop peu d'écho, le rapport Y/HX est supérieur à 1 et la réflectivité augmente ; là où elle en prédit trop, elle diminue. Les directions sont discrétisées tous les $0,5^\circ$, ce qui rend H creuse : chaque impulsion ne voit que les directions proches de son axe. Après 20 itérations, on lit pour chaque direction la distance du maximum de X , affinée par une interpolation gaussienne, et on écarte les directions sans écho net, qui correspondent aux ombres. On en tire le MNT comme précédemment. La figure 4.4 montre une tranche de ce tableau pour le balayage simulé.

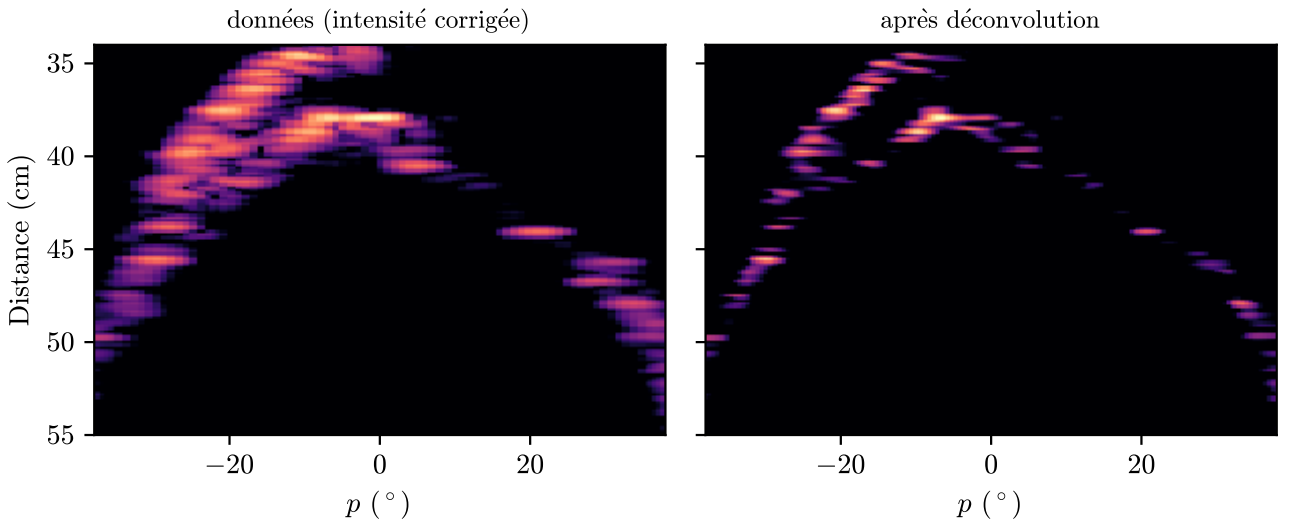


FIGURE 4.4 – Tranche du cube de données le long d'une ligne de balayage qui traverse la conduite : intensité reçue (à gauche) et réflectivité après déconvolution (à droite)

4.5.3 Résultats

Comparaison au relief vrai

Les reliefs reconstruits à partir du balayage piloté sont comparés au relief vrai sur la zone de 44×44 cm entièrement couverte (Fig 4.5 et 4.6). On donne le biais, écart moyen entre le relief reconstruit et le relief vrai, l'écart quadratique moyen (RMS), et la valeur que dépassent 5% des écarts :

Méthode	Biais (mm)	Écart RMS (mm)	95% des écarts sous (mm)
Front montant	15,0	19,0	38,5
Détection classique (barycentre)	3,7	9,8	22,1
Déconvolution	2,6	7,5	16,8

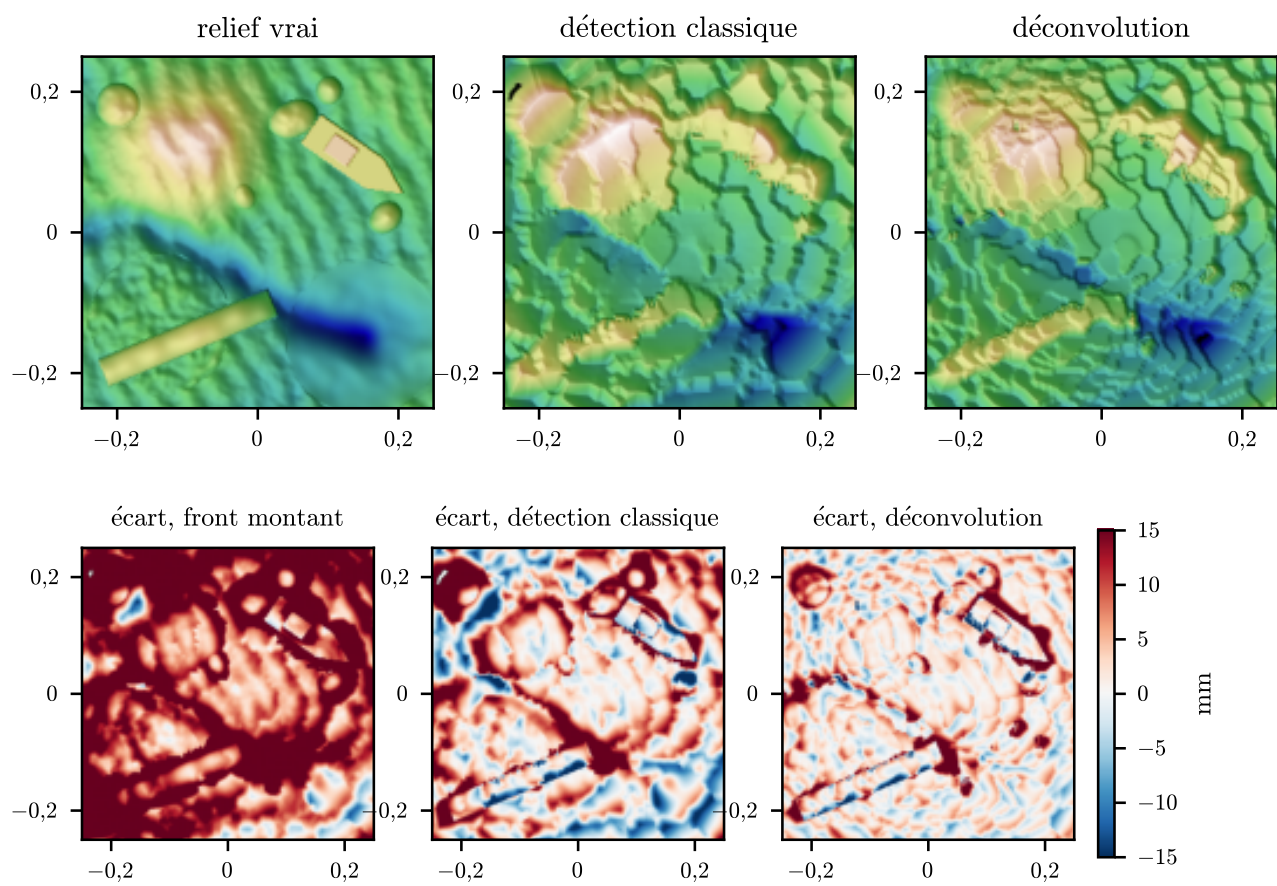


FIGURE 4.5 – Relief vrai et reliefs reconstruits à partir du balayage simulé (en haut) ; écarts au relief vrai (en bas)

Discussion

Le front montant donne le point le plus proche de chaque tache : le fond paraît plus haut qu'il n'est, les objets s'élargissent et le chenal se comble. La détection classique réduit ce biais, mais les objets restent étalés sur la largeur de la tache : la conduite apparaît comme une bosse arrondie, l'épave perd sa forme et ses abords sont surélevés de 1 à 2 cm. La déconvolution rend aux objets leur forme : la conduite retrouve sa largeur et ses flancs, l'épave sa silhouette et le chenal sa continuité. L'écart RMS sur les rochers, la conduite et l'épave passe de 10,5 mm à 7,8 mm. En revanche, les rides de sable, de la taille de la tache, restent à peine visibles et les flancs verticaux de l'épave restent adoucis : la déconvolution améliore la résolution, mais elle ne retrouve pas ce que le faisceau ne distingue pas.

Avec les orientations exactes plutôt qu'estimées, les écarts RMS seraient de 9,7 mm et 7,4 mm : les erreurs de pointage de la sphère ont peu d'influence.

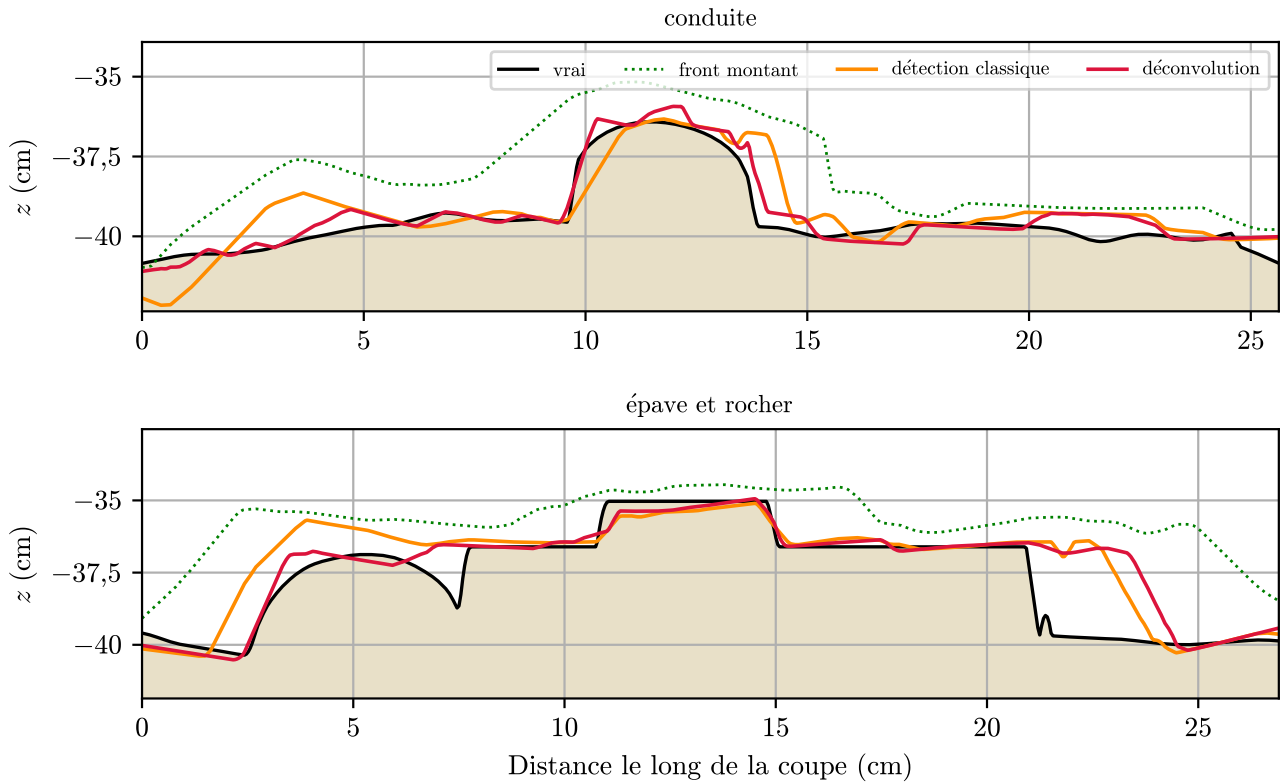


FIGURE 4.6 – Coupes à travers la conduite, puis à travers l'épave et un rocher

4.5.4 Influence des paramètres

Nombre d'itérations

Sans déconvolution, en moyennant simplement les données pondérées par le diagramme, l'écart RMS est de 13,3 mm. Il diminue fortement pendant les dix premières itérations, reste stable jusqu'à vingt, puis augmente très lentement (Fig 4.7) : l'algorithme se met alors à reproduire le chatolement comme s'il s'agissait de détails du fond. Nous nous arrêtons à 20 itérations. Sans chatolement, c'est-à-dire avec des échos égaux à leur moyenne, les écarts seraient de 8,8 mm et 6,1 mm : le chatolement coûte entre un millimètre et un millimètre et demi.

Erreur de pointage

Une erreur de pointage aléatoire, différente à chaque impulsion, a peu d'influence : une erreur de 1° (RMS) ne porte l'écart qu'à 10,3 mm pour la détection classique et 7,4 mm pour la déconvolution (Fig 4.8). Chaque point du fond étant vu par une vingtaine d'impulsions, leurs erreurs se compensent. Une erreur systématique, comme l'orientation initiale, déplace l'image d'un bloc sans la rendre floue.

Fréquence d'émission

L'ouverture du faisceau fixe la qualité des deux méthodes. Avec le même disque, à 2 MHz, les écarts tombent à 6,1 mm et 5,0 mm ; à 500 kHz, ils montent à 17,6 mm et 12,3 mm.

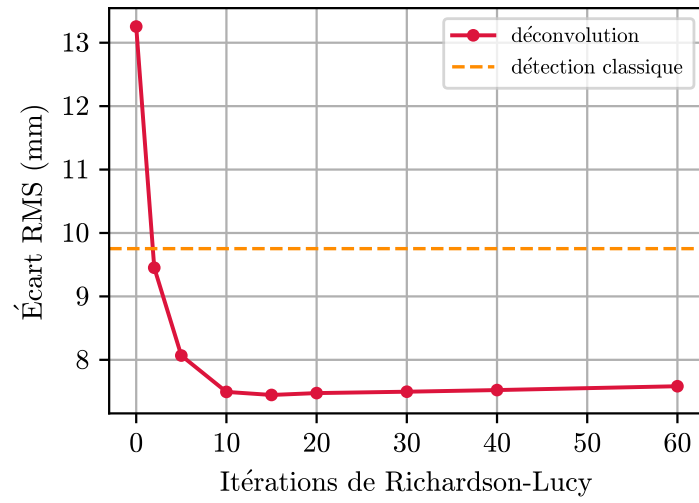


FIGURE 4.7 – Écart RMS en fonction du nombre d'itérations

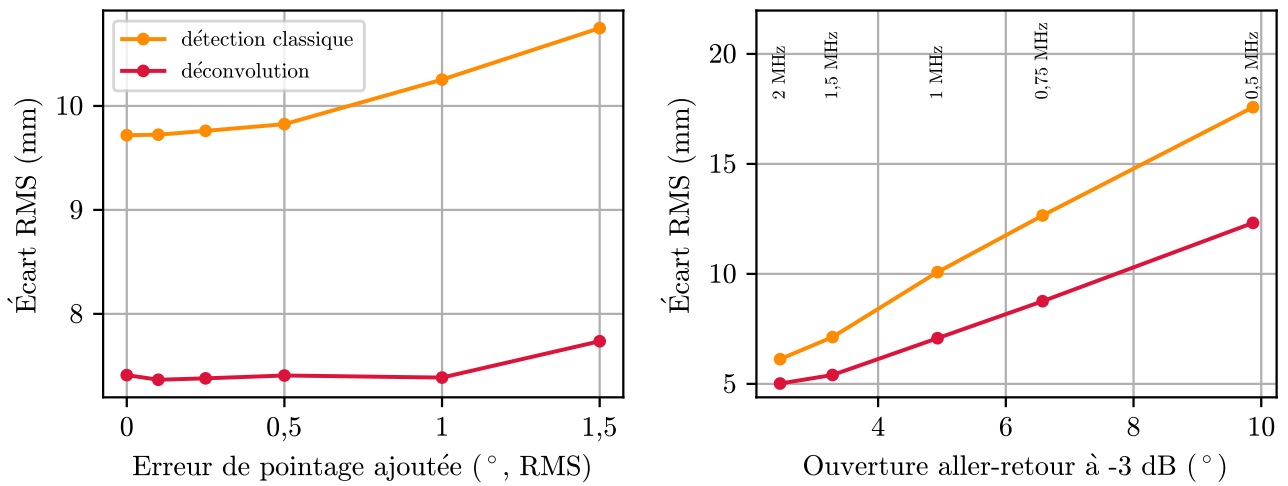


FIGURE 4.8 – Écart RMS en fonction de l'erreur de pointage ajoutée (à gauche) et de l'ouverture du faisceau, pour un même disque à différentes fréquences (à droite)

4.6 Caractérisation des fonds

Mesure de l'indice de rétrodiffusion

D'après (2.12), l'énergie de l'écho d'une impulsion, corrigée de la divergence et de l'absorption, vaut :

$$E = K^2 E_p \sigma(\theta) \frac{r^2 \Psi}{\cos \theta} \quad \text{d'où} \quad \hat{\sigma} = \frac{E \cos \theta}{K^2 E_p r^2 \Psi} \quad (4.4)$$

où K regroupe le niveau d'émission, la sensibilité et le gain de la chaîne, et E_p l'énergie de l'impulsion. La constante K s'étalonne à l'aide d'une cible de référence [23] et l'incidence θ se calcule avec la normale du MNT reconstruit. Chaque impulsion donne ainsi une mesure de l'indice de rétrodiffusion, sous une incidence connue.

Réponse angulaire

Nous avons vu au chapitre 2 que la nature d'un fond se lit dans la façon dont son indice de rétrodiffusion varie avec l'angle d'incidence. Or, depuis le centre de la sphère, chaque point du fond n'est vu que sous un seul angle. Pour mesurer la réponse angulaire d'un même endroit, il faut l'observer depuis plusieurs positions (Fig 4.9) : nous simulons cinq balayages, la sphère étant placée au centre puis aux quatre coins d'un carré de 26 cm de côté.

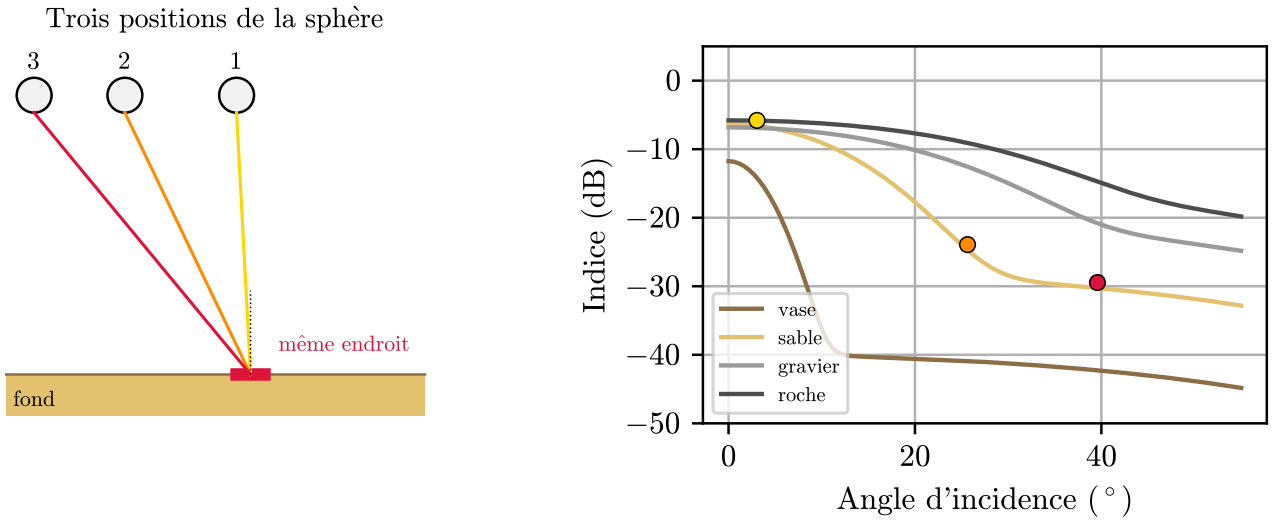


FIGURE 4.9 – Principe de la caractérisation : un même endroit du fond, vu depuis trois positions de la sphère, l'est sous trois incidences (à gauche) ; les trois mesures de l'indice de rétrodiffusion se placent sur la courbe du type de fond (à droite)

L'estimateur moyenne σ sur toute la tache, avec la pondération D^4 : il mesure la courbe du modèle moyennée sur le faisceau, qui ne s'en écarte que près de la verticale. Les mesures, moyennées par tranches d'incidence sur les zones homogènes, suivent ces courbes à quelques décibels près (Fig 4.10) ; de biais, elles sont un peu plus fortes, car le relief (rides, flancs du chenal) incline une partie des facettes vers le transducteur. Les réponses des quatre fonds sont nettement séparées : de biais, la vase renvoie environ 10 dB de moins que le sable, qui renvoie lui-même environ 10 dB de moins que le gravier.

Classification

Le fond est découpé en cases de 3 cm. Dans chaque case, on moyenne les mesures de chaque position, puis on choisit le type de fond dont la courbe passe au plus près des points obtenus. Le sable couvrant 60% des cases, le taux global de réussite serait trompeur : nous donnons la moyenne des taux de réussite des quatre types, qui vaudrait 25% pour un choix au hasard. Elle est de 57% avec la seule position centrale et de 64% avec les cinq positions (Fig 4.11). Le gravier est reconnu dans 78% des cases, le sable dans 67%, la roche dans 61% et la vase dans 48% seulement.

Discussion

Une case ne reçoit que quelques impulsions par position : le chatolement et les pentes du relief dispersent ses mesures de plusieurs décibels. Ainsi, du sable aux rides bien orientées passe pour du gravier, et le chenal, à peine plus large que la tache, prend la réponse du sable qui le borde. La caractérisation est donc fiable à l'échelle des zones, mais pas encore à l'échelle de la tache ; il faudrait davantage de positions ou un faisceau plus étroit.

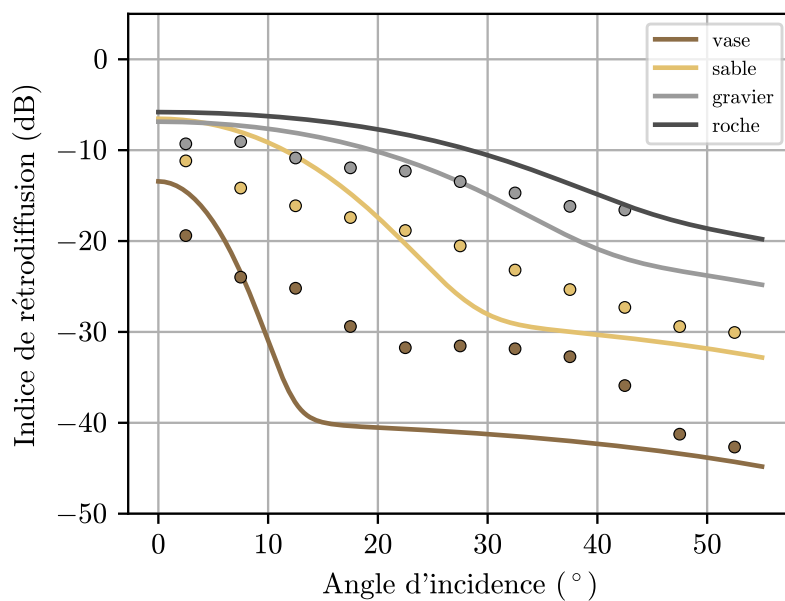


FIGURE 4.10 – Indice de rétrodiffusion mesuré sur les zones homogènes (points, moyenne par tranche de 5°) et modèle moyenné sur le faisceau (traits)

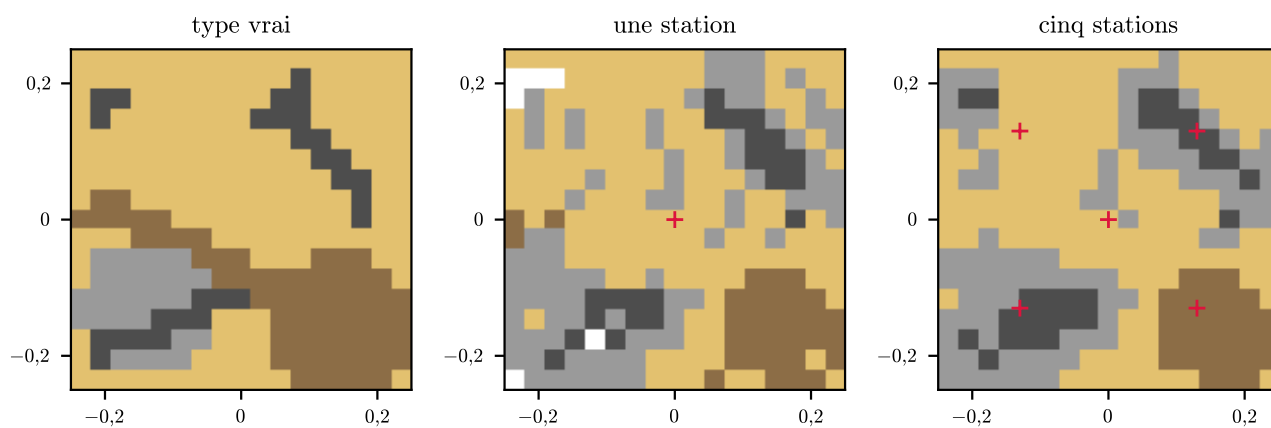


FIGURE 4.11 – Types de fond vrais et retrouvés à partir d'une et de cinq positions de la sphère (croix), par cases de 3 cm

Chapitre 5

Réalisation du système

Ce chapitre décrit le système réel et les programmes qui le piloteraient : les programmes des deux cartes électroniques, le programme de pilotage et d'acquisition, et le programme d'imagerie. Le système n'ayant pas été construit, ces programmes n'ont pas été essayés sur le matériel ; la partie commune des programmes des cartes et le protocole de communication sont vérifiés par des tests sur ordinateur. Nous terminons par un protocole expérimental.

5.1 Architecture

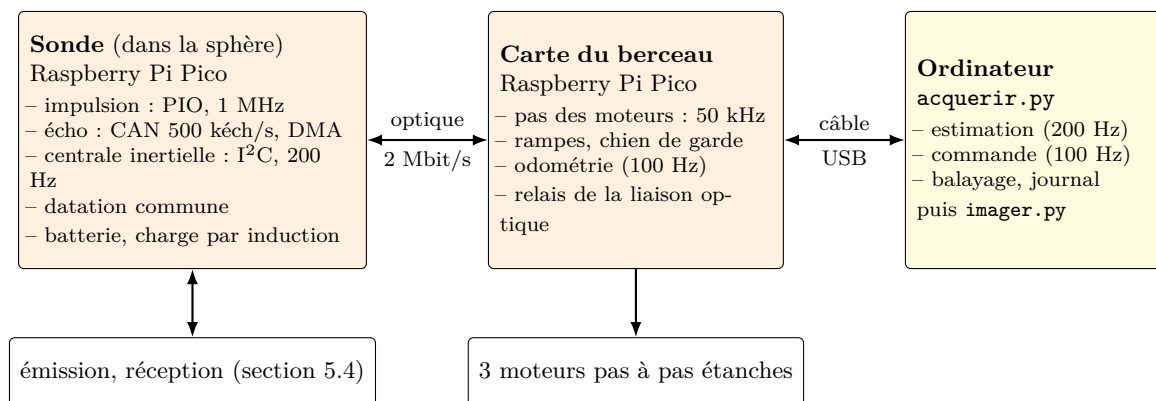


FIGURE 5.1 – Le système réel : la sonde dans la sphère, la carte du berceau et l'ordinateur

Le système comporte deux cartes électroniques et un ordinateur (Fig 5.1). La sonde, dans la sphère, émet les impulsions, échantillonne les échos et lit la centrale inertielle ; elle date toutes ses mesures avec la même horloge et les envoie par la liaison optique. La carte du berceau génère les pas des trois moteurs, renvoie l'odométrie et relaie la liaison optique : un seul câble la relie à l'ordinateur. L'ordinateur estime l'orientation, l'asservit, conduit le balayage et enregistre le journal. Les échanges suivent un protocole binaire commun, défini en C++ (`firmware/commun/protocole.h`) et en Python (`sonar/protocole.py`) : chaque trame commence par deux octets de synchronisation et se termine par un CRC-16.

Message	Sens	Contenu	Fréquence
CONSIGNE_ROUES	ordinateur → berceau	vitesse des trois moteurs (micropas/s), marche	100 Hz
ODOMETRIE	berceau → ordinateur	date, positions et vitesses des moteurs, état	100 Hz
DEMARRER, ARRETER	ordinateur → sonde	cadence, nombre d'échantillons, périodes	
IMU	sonde → ordinateur	date, gyromètre et accéléromètre bruts	200 Hz
PING	sonde → ordinateur	date d'émission, numéro, 576 échantillons	50 Hz

5.2 Carte du berceau

Génération des pas

Un moteur pas à pas tourne d'un angle fixe, un pas, à chaque impulsion électrique qu'il reçoit ; sa vitesse est donc la fréquence de ces impulsions. Le programme de la carte du berceau (`firmware/berceau`, annexe 5)

génère les pas par accumulateur de phase. Une interruption, 50 000 fois par seconde, ajoute à un compteur de 32 bits l'incrément $|v| 2^{32}/50\,000$, où v est la vitesse en micropas par seconde ; chaque débordement du compteur donne un pas. La fréquence des pas vaut ainsi exactement v , sans division dans l'interruption, et la position est comptée au passage : c'est l'odométrie. Une boucle à 1 kHz fait suivre à la vitesse une rampe d'accélération et commande le sens de rotation. Sans consigne pendant 200 ms, la carte arrête les moteurs.

Relais de la liaison optique

Les octets reçus de la sonde sont rangés par une interruption dans une mémoire tampon circulaire, puis découpés en trames et transmis à l'ordinateur ; les ordres de l'ordinateur destinés à la sonde lui sont renvoyés par la liaison optique.

5.3 Sonde

Le programme de la sonde (**firmware/sonde**, annexe 6) utilise plusieurs périphériques du microcontrôleur RP2040 [24] :

- une machine d'état programmable (PIO), petit processeur annexe qui pilote des broches au cycle d'horloge près, génère le train d'impulsions : deux sorties en opposition, huit cycles par période, cadencées à 8 MHz, soit exactement 1 MHz, pendant le nombre de périodes demandé ;
- le convertisseur analogique-numérique, réglé à 500 000 échantillons par seconde, range ses mesures en mémoire par accès direct à la mémoire (DMA), un mécanisme qui copie les données sans occuper le processeur ; il démarre au même instant que l'impulsion, dont la date est enregistrée ;
- le bus I²C lit la centrale inertielle (type MPU-6050 [25]) toutes les 5 ms, réglée à ± 500 °/s et ± 2 g ;
- une liaison série à 2 Mbit/s, transmise par DMA, envoie les messages : une impulsion (1 170 octets) part en 6 ms, pendant que le processeur continue de lire la centrale.

Une impulsion et son enregistrement durent 1,2 ms ; l'impulsion suivante part 20 ms plus tard.

5.4 Chaîne d'émission et de réception

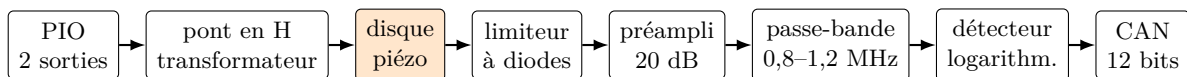


FIGURE 5.2 – Chaîne d'émission et de réception

Émission

Les deux sorties de la PIO commandent un pont en H qui excite le disque, par l'intermédiaire d'un petit transformateur, sous quelques dizaines de volts.

Réception

Un limiteur à diodes protège le préamplificateur pendant l'émission et un filtre passe-bande centré sur 1 MHz limite le bruit (Fig 5.2). Un détecteur logarithmique convertit ensuite l'enveloppe en une tension proportionnelle à son niveau en décibels : il couvre 110 dB sur la pleine échelle du convertisseur, soit 0,03 dB par pas. Il absorbe ainsi sans réglage les 37 dB qui séparent les échos du sable à la verticale et de la vase de biais, là où un détecteur linéaire saturerait sur les uns ou noierait les autres. La simulation reproduit cette chaîne.

Liaison optique

De chaque côté, une diode électroluminescente bleue émet et une photodiode, suivie d'un amplificateur et d'un comparateur, reçoit. À travers la paroi transparente et quelques centimètres d'eau claire, la lumière bleue est très peu absorbée.

5.5 Programmes d'acquisition et d'imagerie

Acquisition

Le programme `acquerir.py` pilote un balayage, sur le système réel ou en simulation :

```
python acquerir.py --port /dev/ttyACM0 --sortie balayage.npz
python acquerir.py --simulation maquette.npz --sortie balayage.npz
```

Il démarre la sonde, initialise l'estimation au repos pendant 0,5 s (verticale et biais du gyromètre), puis exécute la boucle de commande à 100 Hz : lire les messages reçus, traiter les mesures de la centrale, calculer la vitesse angulaire, la convertir en vitesses des moteurs et l'envoyer. Les impulsions sont enregistrées telles quelles dans le journal, avec les mesures brutes, l'odométrie, les consignes, l'historique de l'orientation estimée et la configuration du dispositif.

Imagerie

Le programme `imager.py` lit un ou plusieurs journaux, interpole l'orientation estimée à l'instant de chaque impulsion, puis produit les deux MNT, les indices de rétrodiffusion et, à partir de plusieurs positions, la classification des fonds :

```
python imager.py balayage.npz --sortie images/  
python imager.py st1.npz st2.npz st3.npz st4.npz st5.npz --classer
```

Avec l'option `--relief`, il compare le résultat au relief vrai : c'est ainsi qu'ont été obtenus les résultats du chapitre 4.

Tests

Les tests (`python -m unittest`) vérifient la physique (célérité, absorption, directivité), la cinématique de la sphère, le filtre d'estimation, un petit balayage piloté et une petite image. Un test compile la partie commune des programmes des cartes sur l'ordinateur, vérifie la génération des pas et relit en Python les trames écrites par le code C.

5.6 Protocole expérimental proposé

Matériel

- Bassin vitré de 80×80 cm, rempli sur 55 cm ;
- Sphère en acrylique de 100 mm, avec la sonde, le disque piézoélectrique et la centrale inertielle ;
- Berceau, trois roues omnidirectionnelles et trois moteurs pas à pas étanches, carte du berceau ;
- Relief imprimé en quatre tuiles, sable, gravier et argile pour la vase ;
- Thermomètre, plaque plane, bille d'acier suspendue ;
- Ordinateur.

Protocole

1. Mesurer la température de l'eau, d'où la célérité (2.2).
2. Vérifier l'effort qui plaque la sphère contre les roues en mesurant sa poussée nette, et l'ajuster par lestage.
3. Étalonner l'accéléromètre en immobilisant la sphère dans une vingtaine d'orientations ; mesurer le biais du gyromètre au repos.
4. Repérer l'orientation initiale par une marque sur la sphère, alignée sur la première roue.
5. Étalonner les distances : le faisceau pointé vers une plaque plane posée à une hauteur connue donne le décalage de temps entre l'émission et l'échantillonnage.
6. Mesurer le diagramme et la constante K de la chaîne en balayant autour de la bille d'acier, dont l'écho se calcule [23].
7. Poser le relief imprimé au fond du bassin, recouvrir les zones de sable, de gravier et de vase, et le balayer ; comparer le MNT au fichier STL.
8. Recommencer depuis plusieurs positions du berceau pour la caractérisation.

Chapitre 6

Transposition en mer

Au vu des dimensions de la maquette, il n'est pas possible de transposer directement les résultats à un fond réel. Ce chapitre en donne des ordres de grandeur ; il ne s'agit pas d'un dimensionnement.

6.1 Similitude

Pour qu'une maquette se comporte comme le fond réel, il faut conserver les rapports des longueurs à la longueur d'onde : la même valeur de ka donne le même faisceau, et le même rapport entre la taille des détails et la tache. Au 1/50, les 40 cm du bassin deviennent 20 m et 1 MHz devient 20 kHz ; le disque mesurerait alors 64 cm de diamètre et la tache 1,7 m à 20 m. Deux grandeurs ne suivent pas l'échelle : l'absorption, faible à 20 kHz en mer, et la taille des grains du sédiment. Le sable du bassin, vu par une onde de 1,5 mm, se comporte comme des galets de plusieurs centimètres vus par une onde de 7,5 cm : la maquette reproduit la géométrie du fond, pas sa nature.

6.2 Choix de la fréquence

L'absorption aller-retour croît rapidement avec la fréquence (Fig 6.1) : par 100 m de fond, elle vaut 1 dB à 20 kHz et 11 dB à 200 kHz ; par 500 m, 3 dB et 54 dB. On choisit donc la fréquence la plus élevée compatible avec la profondeur, puisque c'est elle qui donne la meilleure résolution pour une taille de transducteur donnée.

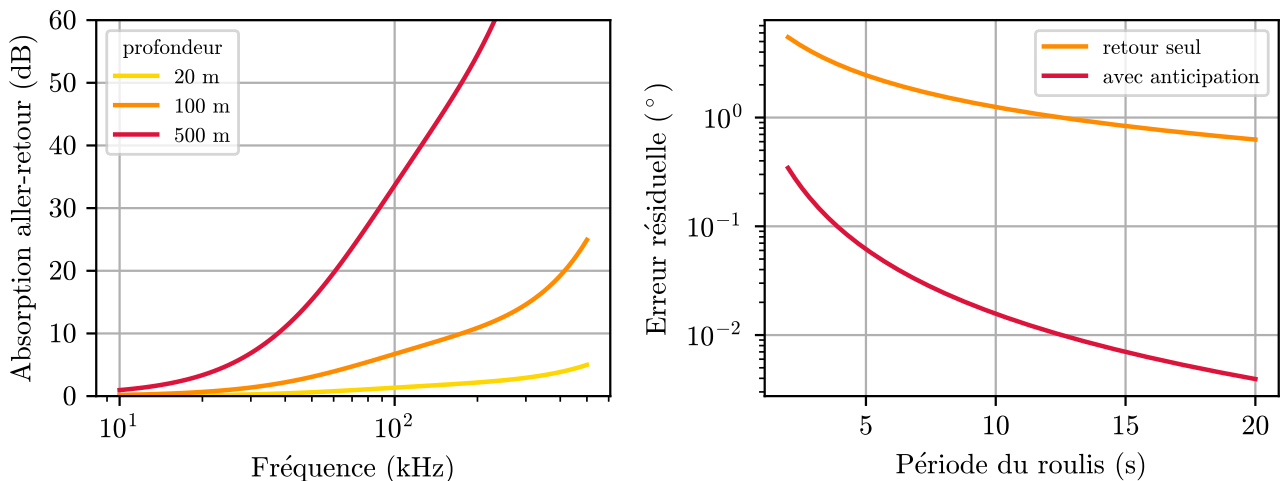


FIGURE 6.1 – Absorption aller-retour en mer à 10 °C en fonction de la fréquence (à gauche) et erreur résiduelle de pointage pour un roulis de 10° en fonction de sa période (à droite)

6.3 Stabilisation du faisceau

Sur un navire, le roulis incline le sondeur. Fixée sous la coque, la sphère peut maintenir son faisceau dans la direction voulue : la commande reste la même, la vitesse angulaire de la coque devenant une perturbation. Pour un roulis de 10° d'amplitude et de 8 s de période, la seule correction de l'erreur laisse une erreur de $1,6^\circ$. Si l'on ajoute à la consigne la vitesse de la coque, mesurée par le gyromètre, il ne reste que l'effet d'un retard de 20 ms, soit $0,02^\circ$ (Fig 6.1). En mer, cependant, l'accéléromètre mesure aussi les accélérations du navire et ne donne plus la verticale : il faudrait utiliser la centrale d'attitude du navire.

6.4 Tenue à la pression

Sous la pression p , la paroi d'une sphère de rayon R et d'épaisseur e travaille uniformément, avec une contrainte $pR/(2e)$, deux fois plus faible que dans un cylindre de même rayon. À 100 m de profondeur (1 MPa), une sphère de 5 cm de rayon en acrylique, qui supporte de l'ordre de 20 MPa, n'a besoin que d'un peu plus d'un millimètre d'épaisseur, sous réserve du flambage. La poussée nette, qui plaque la sphère contre les roues, ne dépend pas de la profondeur tant que la paroi ne se déforme pas.

6.5 Réfraction

En mer, la célérité varie avec la profondeur et les rayons sonores se courbent : la position du point visé se calcule en suivant le rayon à travers le profil de célérité, mesuré par une sonde. Dans le bassin, ce n'était pas nécessaire.

Chapitre 7

Bilan, limites et suite du travail

Ce chapitre résume ce qui a été fait, ce qui ne l'a pas été, les limites des résultats présentés et ce qu'il reste à faire pour aboutir à un dispositif réel.

7.1 Ce qui a été fait

- L'étude de la physique de la mesure : propagation, transducteur, résolution, rétrodiffusion par le fond, équation du sonar, chatolement, et ce que mesure réellement un sonar monofaisceau.
- La conception d'un dispositif d'orientation par une sphère immergée et l'étude de sa mécanique : équilibre dans l'eau, entraînement par trois roues et choix de leur position, mesure de l'orientation par une centrale inertielle et asservissement.
- Un simulateur complet : création de reliefs, modélisation de l'eau, des fonds et du bassin, calcul des échos avec leurs fluctuations, simulation des moteurs, de la centrale inertielle et des liaisons.
- Les programmes de pilotage, d'acquisition et d'imagerie, qui ont fonctionné sur les données simulées : la détection classique du fond, une méthode de reconstruction par déconvolution et une caractérisation des fonds par leur réponse angulaire.
- Les programmes des deux cartes électroniques et un protocole expérimental.

7.2 Ce qui n'a pas été fait

- Le dispositif n'a pas été construit et aucune mesure n'a été réalisée : tous les résultats sont issus de simulations.
- Les programmes des cartes n'ont pas été compilés pour le microcontrôleur ni essayés sur le matériel ; seule leur partie commune (protocole, génération des pas) a été compilée et testée sur ordinateur.
- L'électronique n'est décrite que par ses fonctions : le générateur d'impulsions, la chaîne de réception, la liaison optique et la charge par induction n'ont fait l'objet ni d'un schéma détaillé ni d'un choix de composants.
- La mécanique du berceau (boîtiers étanches, accouplements magnétiques, réglage de la position des roues) n'a pas été dessinée.
- Plusieurs grandeurs sont supposées et non mesurées : le niveau d'émission et la sensibilité du transducteur, les paramètres de rétrodiffusion des quatre fonds, le coefficient de frottement entre les roues et la sphère mouillée, le moment d'inertie de la sphère et les défauts de la centrale inertielle.

7.3 Limites de l'étude

La simulation et l'imagerie reposent sur le même modèle

La déconvolution utilise exactement le diagramme du transducteur et la forme de l'impulsion qui ont servi à simuler les données. Sur le dispositif réel, ils ne seraient connus qu'à la précision de leur mesure, et le gain apporté par la déconvolution serait plus faible. Les écarts obtenus en simulation sont donc optimistes ; seule l'expérience dira ce qu'il en reste.

Ce que le simulateur ne prend pas en compte

Les échos multiples (fond, surface, puis fond à nouveau), les échos des parois du bassin, les lobes secondaires à plus de 16° de l'axe, la zone morte qui suit l'émission, la diffusion multiple entre objets et les surplombs, le fond étant décrit par un champ de hauteurs. La célérité est supposée uniforme et les rayons rectilignes. La rétrodiffusion repose sur un modèle empirique, appliqué maille par maille à des diffuseurs indépendants.

La mécanique est idéalisée

Les moteurs suivent exactement leurs consignes, sans perte de pas ; le frottement est constant et le glissement des roues est représenté par un simple facteur ; les vibrations et les défauts des galets sont ignorés.

Les chiffres dépendent du fond choisi

Les écarts entre relief reconstruit et relief vrai valent pour la maquette de référence, pour une seule réalisation aléatoire du chatolement et des défauts de mesure. Un fond plus accidenté ou plus lisse donnerait d'autres valeurs ; c'est l'écart entre les deux méthodes, plus que leurs valeurs absolues, qu'il faut retenir.

La caractérisation reste incertaine

La classification case par case ne dépasse pas 64% de réussite en moyenne sur les quatre types de fond, et les courbes de rétrodiffusion utilisées sont des valeurs typiques et non des mesures.

La maquette ne reproduit pas tout

Comme on l'a vu au chapitre 6, la maquette respecte la géométrie du fond rapportée à la longueur d'onde, mais pas la nature des sédiments, ni l'absorption en mer.

7.4 Ce qu'il reste à faire

1. Construire la sphère, le berceau et les deux cartes, compiler et mettre au point leurs programmes.
2. Mesurer le diagramme du transducteur, son niveau d'émission et sa sensibilité, et utiliser le diagramme mesuré dans l'imagerie à la place du modèle.
3. Balayer le relief imprimé et comparer le relief reconstruit au fichier qui a servi à l'imprimer : c'est la vraie mesure de la fidélité, et la validation du simulateur.
4. Mesurer les courbes de rétrodiffusion de bacs de sable, de gravier et d'argile, et classer les fonds à partir de ces courbes mesurées.
5. Améliorer l'imagerie : déconvoluer ensemble les balayages faits depuis plusieurs positions, favoriser un fond continu, réduire le chatolement en émettant à plusieurs fréquences.
6. Pour un usage en mer : utiliser la centrale d'attitude du navire, tenir compte du profil de célérité, et dimensionner un boîtier résistant à la pression.

Conclusion

Reprendre ce travail commencé en 2020 m'a permis de le mener jusqu'à un dispositif complet, décrit, programmé et simulé. Il reste maintenant à le construire.

Pour orienter le faisceau, une sphère immergée qui porte le transducteur et que font tourner trois roues remplace avantageusement une tourelle à deux axes : tous les faisceaux partent de son centre, il n'y a ni direction singulière ni déplacement du transducteur, l'eau n'exerce sur elle aucun couple et sa poussée nette suffit à la plaquer contre les roues. Placées à $35,26^\circ$ du pôle, les roues ont des axes orthogonaux et commandent chacune une composante de la rotation. Une centrale inertielle placée au centre, recalée à chaque arrêt grâce au compte des pas, donne la direction du faisceau au dixième de degré.

Pour répondre à la problématique, la simulation montre que l'on restitue d'autant plus fidèlement le fond que l'on exploite tout l'écho et non une seule distance par impulsion : en traitant les impulsions comme un cube de données rendu flou par un faisceau connu, et en le déconvoluant, l'écart au relief vrai passe de 9,8 mm à 7,5 mm et les objets retrouvent leur forme. Les détails plus petits que la tache restent cependant hors d'atteinte : seul un faisceau plus étroit permettrait de les distinguer. La réponse angulaire de la rétrodiffusion, mesurée depuis plusieurs positions, distingue la vase, le sable, le gravier et la roche à l'échelle des zones ; case par case, la classification reste incertaine.

Ces résultats sont issus de simulations qui reposent sur des modèles et des grandeurs supposées (chapitre 7) : ils restent à vérifier par l'expérience. La prochaine étape est de construire la sphère, son berceau et les deux cartes, d'imprimer le relief et de comparer les mesures à la simulation en suivant le protocole proposé.

Bibliographie

- [1] Xavier LURTON. *An Introduction to Underwater Acoustics : Principles and Applications*. Springer et Praxis Publishing (2002).
- [2] Adrien MOURS. “Localisation de cible en sonar actif”. Thèse de doctorat, Université Grenoble Alpes (2017).
- [3] Lawrence E. KINSLER, Austin R. FREY, Alan B. COPPENS et James V. SANDERS. *Fundamentals of Acoustics*. 4^e édition, Wiley (2000).
- [4] Michel LEGRIS, Katia LEBART, Fabrice FOHANNO et Benoît ZERR. “Les capteurs d’imagerie en robotique sous-marine : tendances actuelles et futures”. In : *Traitement du Signal* 20.2 (2003).
- [5] Herman MEDWIN. “Speed of sound in water : a simple equation for realistic parameters”. In : *Journal of the Acoustical Society of America* 58.6 (1975), p. 1318-1319.
- [6] R. E. FRANCOIS et G. R. GARRISON. “Sound absorption based on ocean measurements. Part II : Boric acid contribution and equation for total absorption”. In : *Journal of the Acoustical Society of America* 72.6 (1982), p. 1879-1890.
- [7] Carl F. EYRING. “Reverberation time in ‘dead’ rooms”. In : *Journal of the Acoustical Society of America* 1.2A (1930), p. 217-241.
- [8] Robert J. URICK. *Principles of Underwater Sound*. 3^e édition, McGraw-Hill (1983).
- [9] Applied Physics Laboratory, University of Washington. *APL-UW High-Frequency Ocean Environmental Acoustic Models Handbook*. Rapport APL-UW TR 9407 (1994).
- [10] Darrell R. JACKSON et Michael D. RICHARDSON. *High-Frequency Seafloor Acoustics*. Springer (2007).
- [11] Kenneth V. MACKENZIE. “Bottom reverberation for 530- and 1030-cps sound in deep water”. In : *Journal of the Acoustical Society of America* 33.11 (1961), p. 1498-1504.
- [12] Joseph W. GOODMAN. “Some fundamental properties of speckle”. In : *Journal of the Optical Society of America* 66.11 (1976), p. 1145-1150.
- [13] John SIMMONDS et David MACLENNAN. *Fisheries Acoustics : Theory and Practice*. 2^e édition, Blackwell (2005).
- [14] Tom B. LAUWERS, George A. KANTOR et Ralph L. HOLLIS. “A dynamically stable single-wheeled mobile robot with inverse mouse-ball drive”. In : *Proceedings of the IEEE International Conference on Robotics and Automation* (2006), p. 2884-2889.
- [15] Masaaki KUMAGAI et Takaya OCHIAI. “Development of a robot balancing on a ball”. In : *International Conference on Control, Automation and Systems* (2008), p. 433-438.
- [16] Ken SHOEMAKE. “Animating rotation with quaternion curves”. In : *Computer Graphics (SIGGRAPH '85)* 19.3 (1985), p. 245-254.
- [17] Robert MAHONY, Tarek HAMEL et Jean-Michel PFLIMLIN. “Nonlinear complementary filters on the special orthogonal group”. In : *IEEE Transactions on Automatic Control* 53.5 (2008), p. 1203-1218.
- [18] Eric FOXLIN. “Pedestrian tracking with shoe-mounted inertial sensors”. In : *IEEE Computer Graphics and Applications* 25.6 (2005), p. 38-46.
- [19] John T.-Y. WEN et Kenneth KREUTZ-DELGADO. “The attitude control problem”. In : *IEEE Transactions on Automatic Control* 36.10 (1991), p. 1148-1162.
- [20] Dietmar SAUPE. “Algorithms for random fractals”. In : H.-O. PEITGEN et D. SAUPE (dir.), *The Science of Fractal Images*, Springer (1988), p. 71-136.
- [21] William H. RICHARDSON. “Bayesian-based iterative method of image restoration”. In : *Journal of the Optical Society of America* 62.1 (1972), p. 55-59.
- [22] Leon B. LUCY. “An iterative technique for the rectification of observed distributions”. In : *The Astronomical Journal* 79.6 (1974), p. 745-754.
- [23] Kenneth G. FOOTE, Hans P. KNUDSEN, Gunnar VESTNES, David N. MACLENNAN et E. John SIMMONDS. “Calibration of acoustic instruments for fish density estimation : a practical guide”. *ICES Cooperative Research Report* 144 (1987).
- [24] Raspberry Pi Ltd. *RP2040 Datasheet : A microcontroller by Raspberry Pi*. Documentation technique.
- [25] InvenSense. *MPU-6000 and MPU-6050 Register Map and Descriptions*. Révision 4.2 (2013).

Annexes

Les programmes complets se trouvent dans le dossier `programmes`, avec un fichier `README.md` qui explique comment les utiliser. Nous reproduisons ici les principaux.

Annexe 1 : simulation de l'écho

```
1 from __future__ import annotations
2
3 import numpy as np
4 from scipy.ndimage import map_coordinates
5 from scipy.spatial import cKDTree
6
7 from .milieu import FONDS, Bassin
8 from .relief import Relief
9 from .sphere import faisceau
10 from .transducteur import Recepteur, Transducteur, directivite_piston
11
12
13 class SimulateurSonar:
14     def __init__(self, relief: Relief, bassin: Bassin, transducteur: Transducteur, recepteur: Recepteur,
15                 R_sphere: float = 0.050, psi_max: float | None = None, graine: int = 0, sur_echantillonnage: int
16                 ↪ = 8,
17                 lobes_secondaires: float = 2.0, centre=(0.0, 0.0, 0.0)):
18         self.relief = relief
19         self.centre = np.asarray(centre, dtype=float)
20         self.bassin = bassin
21         self.tr = transducteur
22         self.rx = recepteur
23         self.R = R_sphere
24         self.c = bassin.eau.c
25         self.alpha = bassin.eau.alpha(transducteur.f0)
26         self.k = 2 * np.pi * transducteur.f0 / self.c
27         self.ka = transducteur.ka(self.c)
28         self.psi_max = psi_max if psi_max is not None else lobes_secondaires * transducteur.premier_zero(self.c)
29         self.rng = np.random.default_rng(graine)
30         self.sur = sur_echantillonnage
31
32         X, Y = relief.maillage()
33         self.P = np.stack([X.ravel(), Y.ravel(), relief.z.ravel()], axis=1)
34         self.N = relief.normales().reshape(-1, 3)
35         self.A = relief.aire().ravel()
36         cl = relief.classe.ravel()
37         self.R0_2 = np.array([f.R0**2 for f in FONDS])[cl]
38         self.d2 = np.array([f.delta**2 for f in FONDS])[cl]
39         self.mu = np.array([10 ** (f.mu_dB / 10) for f in FONDS])[cl]
40         self.visible = self.visibilite(self.centre).ravel()
41         rel = self.P - self.centre
42         self.U0 = rel / np.linalg.norm(rel, axis=1, keepdims=True)
43         self.arbre = cKDTree(self.U0)
44         self.xi = (self.rng.standard_normal(len(self.P)) + 1j * self.rng.standard_normal(len(self.P))) /
45         ↪ np.sqrt(2)
46
47         self.decalage = transducteur.duree
48         fe_fin = recepteur.fe * self.sur
49         self.n_fin = recepteur.n * self.sur
50         tk = np.arange(-4 * self.tr.duree, 4 * self.tr.duree + 1 / fe_fin, 1 / fe_fin)
51         self.noyau = self.tr.enveloppe(tk)
52         self.noyau_centre = len(tk) // 2
53         self.K = 10 ** ((transducteur.SL + transducteur.sensibilite + recepteur.gain_dB) / 20)
54         self.v_bruit = recepteur.tension_bruit() * 10 ** (recepteur.gain_dB / 20)
55
56     def visibilite(self, origine, n_pas: int = 64, marge: float = 5e-4):
57         r = self.relief
58         P = self.P - origine
59         vis = np.ones(len(P), dtype=bool)
60         for s in np.linspace(0.05, 0.98, n_pas):
61             Q = origine + s * P
62             i = (Q[:, 1] - r.y[0]) / r.h
63             j = (Q[:, 0] - r.x[0]) / r.h
64             zt = map_coordinates(r.z, [i, j], order=1, mode="nearest")
65             vis &= zt < Q[:, 2] + marge
```

```

64         return vis.reshape(r.z.shape)
65
66     def sigma(self, idx, cos_inc):
67         c = np.clip(cos_inc, 1e-3, 1.0)
68         c2 = c * c
69         tan2 = (1 - c2) / c2
70         kir = self.RO_2[idx] / (8 * np.pi * self.d2[idx] * c2 * c2) * np.exp(-tan2 / (2 * self.d2[idx]))
71         return kir + self.mu[idx] * c2
72
73     def facettes(self, b):
74         corde = 2 * np.sin(min(self.psi_max * 1.15, np.pi / 2) / 2)
75         return np.asarray(self.arbre.query_ball_point(b, corde), dtype=np.intp)
76
77     def signal(self, q, coherent: bool = True):
78         b = faisceau(q)
79         face = self.centre + self.R * b
80         idx = self.facettes(b)
81         idx = idx[self.visible[idx]]
82         d = self.P[idx] - face
83         r = np.linalg.norm(d, axis=1)
84         u = d / r[:, None]
85         cos_psi = np.clip(u @ b, -1, 1)
86         garde = cos_psi > np.cos(self.psi_max)
87         cos_inc = -np.sum(u * self.N[idx], axis=1)
88         garde &= cos_inc > 0
89         idx, r, cos_psi, cos_inc = idx[garde], r[garde], cos_psi[garde], cos_inc[garde]
90         D = directivite_piston(self.ka * np.sqrt(1 - cos_psi**2))
91         amp = self.K * D**2 * np.sqrt(self.sigma(idx, cos_inc) * self.A[idx]) / r**2 * 10 ** (-self.alpha * r /
92         ↪ 10)
93         t = (2 * r / self.c + self.decalage) * self.rx.fe * self.sur
94         i0 = np.floor(t).astype(np.intp)
95         f = t - i0
96         dedans = (i0 >= 0) & (i0 + 1 < self.n_fin)
97         if coherent:
98             w = amp * self.xi[idx] * np.exp(-2j * self.k * r)
99             s = np.zeros(self.n_fin, dtype=complex)
100             for decal, poids in ((0, 1 - f), (1, f)):
101                 s.real += np.bincount(i0[dedans] + decal, (w.real * poids)[dedans], self.n_fin)
102                 s.imag += np.bincount(i0[dedans] + decal, (w.imag * poids)[dedans], self.n_fin)
103             return np.convolve(s, self.noyau, mode="full")[self.noyau_centre:self.noyau_centre + self.n_fin]
104         e = np.zeros(self.n_fin)
105         for decal, poids in ((0, 1 - f), (1, f)):
106             e += np.bincount(i0[dedans] + decal, (amp**2 * poids)[dedans], self.n_fin)
107         return np.convolve(e, self.noyau**2, mode="full")[self.noyau_centre:self.noyau_centre + self.n_fin]
108
109     def ping(self, q, coherent: bool = True):
110         s = self.signal(q, coherent)
111         if coherent:
112             bruit = self.rng.standard_normal(self.n_fin) + 1j * self.rng.standard_normal(self.n_fin)
113             bruit = np.convolve(bruit, self.noyau, mode="same")
114             bruit *= self.v_bruit / np.sqrt(np.mean(np.abs(bruit) ** 2))
115             v = np.abs(s + bruit)
116         else:
117             v = np.sqrt(s + self.v_bruit**2)
118         v = v.reshape(self.rx.n, self.sur).mean(axis=1)
119         code = self.rx.code(v) + self.rng.standard_normal(self.rx.n) * self.rx.bruit_can
120         return np.clip(np.round(code), 0, self.rx.pleine_echelle).astype(np.uint16)
121
122     def portees(self):
123         return self.rx.portees(self.c, self.decalage)

```

Annexe 2 : imagerie

```
1 from __future__ import annotations
2
3 from dataclasses import dataclass
4
5 import numpy as np
6 from scipy import sparse
7 from scipy.interpolate import griddata
8 from scipy.ndimage import convolve1d
9 from scipy.spatial import cKDTree
10
11 from .sphere import direction_pq
12 from .transducteur import directivite_piston
13
14
15 def intensite(codes, recepteur, portees, alpha):
16     v = recepteur.tension(codes)
17     r = np.maximum(portees, 1e-3)
18     return v**2 * r**4 * 10 ** (alpha * r / 5)
19
20
21 def detection_fond(I, portees, seuil_dB=-6.0, r_min=0.15, garde=0.004):
22     I = np.atleast_2d(I)
23     valide = portees > r_min
24     J = np.where(valide, I, 0.0)
25     imax = np.argmax(J, axis=1)
26     pic = J[np.arange(len(J)), imax]
27     seuil = pic * 10 ** (seuil_dB / 10)
28     r = np.full(len(J), np.nan)
29     for k in range(len(J)):
30         if pic[k] <= 0:
31             continue
32         a = imax[k]
33         while a > 0 and J[k, a - 1] >= seuil[k]:
34             a -= 1
35         b = imax[k]
36         while b < J.shape[1] - 1 and J[k, b + 1] >= seuil[k]:
37             b += 1
38         w = J[k, a:b + 1]
39         r[k] = np.sum(w * portees[a:b + 1]) / np.sum(w)
40     return r
41
42
43 def premier_echo(I, portees, seuil_dB=-10.0, r_min=0.15):
44     I = np.atleast_2d(I)
45     J = np.where(portees > r_min, I, 0.0)
46     pic = J.max(axis=1, keepdims=True)
47     au-dessus = J >= pic * 10 ** (seuil_dB / 10)
48     i = np.argmax(au-dessus, axis=1)
49     return np.where(pic[:, 0] > 0, portees[i], np.nan)
50
51
52 def points(directions, distances, R_sphere):
53     return np.asarray(directions) * (R_sphere + np.asarray(distances))[:, None]
54
55
56 def mnt(points_xyz, x, y, rayon_max=0.02):
57     ok = np.all(np.isfinite(points_xyz), axis=1)
58     P = points_xyz[ok]
59     X, Y = np.meshgrid(x, y)
60     Z = griddata(P[:, :2], P[:, 2], (X, Y), method="linear")
61     d, _ = cKDTree(P[:, :2]).query(np.stack([X.ravel(), Y.ravel()], axis=1))
62     Z[d.reshape(X.shape) > rayon_max] = np.nan
63     return Z
64
65
66 @dataclass
67 class Grille:
68     demi_largeur: float
69     pas: float
70
71     def __post_init__(self):
72         n = int(round(2 * self.demi_largeur / self.pas)) + 1
73         self.p = np.linspace(-self.demi_largeur, self.demi_largeur, n)
74         self.q = self.p.copy()
75         P, Q = np.meshgrid(self.p, self.q)
```

```

76         self.forme = P.shape
77         self.directions = direction_pq(P.ravel(), Q.ravel())
78         tau = np.hypot(P, Q).ravel()
79         self.angle_solide = self.pas**2 * np.where(tau > 1e-9, np.sin(tau) / np.where(tau > 1e-9, tau, 1), 1.0)
80
81
82     class Deconvolution:
83     def __init__(self, axes, grille: Grille, ka: float, portees, impulsions, psi_max: float,
84                 R_sphere: float = 0.0, distance_nominale=None):
85         self.grille = grille
86         self.portees = portees
87         U = grille.directions
88         arbre = cKDTree(U)
89         lignes, colonnes, valeurs = [], [], []
90         corde = 2 * np.sin(psi_max / 2)
91         for i, b in enumerate(np.asarray(axes)):
92             j = np.asarray(arbre.query_ball_point(b, corde), dtype=np.intp)
93             if len(j) == 0:
94                 continue
95             if R_sphere > 0 and distance_nominale is not None:
96                 rn = distance_nominale[j][:, None]
97                 v = U[j] * (R_sphere + rn) - R_sphere * b
98                 v /= np.linalg.norm(v, axis=1, keepdims=True)
99                 cos = v @ b
100             else:
101                 cos = U[j] @ b
102                 s = np.sqrt(np.clip(1 - cos**2, 0, 1))
103                 lignes.append(np.full(len(j), i))
104                 colonnes.append(j)
105                 valeurs.append(directivite_piston(ka * s) ** 4 * grille.angle_solide[j])
106         self.A = sparse.csr_matrix((np.concatenate(valeurs), (np.concatenate(lignes), np.concatenate(colonnes))),
107                                   shape=(len(axes), len(U)))
108         self.At = self.A.T.tocsr()
109         self.P = np.asarray(impulsions, dtype=float)
110         self.P /= self.P.sum()
111
112     def _conv(self, Z, transpose=False):
113         noyau = self.P[:, :-1] if transpose else self.P
114         return convolve2d(Z, noyau, axis=1, mode="constant")
115
116     def direct(self, X):
117         return self.A @ self._conv(X)
118
119     def adjoint(self, Y):
120         return self._conv(self.At @ Y, transpose=True)
121
122     def richardson_lucy(self, Y, iterations=30, fond=None, X0=None, rappel=None):
123         Y = np.maximum(Y, 0)
124         b = np.percentile(Y, 5) if fond is None else fond
125         norme = self.adjoint(np.ones_like(Y))
126         norme = np.maximum(norme, 1e-12 * norme.max())
127         X = self.adjoint(Y) / norme if X0 is None else X0.copy()
128         for it in range(iterations):
129             X *= self.adjoint(Y / (self.direct(X) + b)) / norme
130             if rappel is not None:
131                 rappel(it, X)
132         return X
133
134     def retroprojection(self, Y):
135         norme = self.adjoint(np.ones_like(Y))
136         return self.adjoint(Y) / np.maximum(norme, 1e-12 * norme.max())
137
138
139     def surface(X, portees, contraste_min=20.0):
140         n = np.argmax(X, axis=1)
141         j = np.arange(len(X))
142         pic = X[j, n]
143         tiny = 1e-30
144         a = np.log(np.maximum(X[j, np.clip(n - 1, 0, X.shape[1] - 1)], tiny))
145         b = np.log(np.maximum(pic, tiny))
146         c = np.log(np.maximum(X[j, np.clip(n + 1, 0, X.shape[1] - 1)], tiny))
147         den = a - 2 * b + c
148         dn = np.where(den < 0, 0.5 * (a - c) / np.where(den < 0, den, -1), 0.0)
149         dr = portees[1] - portees[0]
150         r = portees[n] + np.clip(dn, -1, 1) * dr
151         fond = np.median(X, axis=1)

```

```

152     r[~(pic > contraste_min * np.maximum(fond, 1e-30))] = np.nan
153     return r, pic
154
155
156 def reflectivite(X, portees, r_surface, demi_fenetre=3):
157     dr = portees[1] - portees[0]
158     out = np.full(len(X), np.nan)
159     ok = np.isfinite(r_surface)
160     n = np.zeros(len(X), dtype=int)
161     n[ok] = np.round((r_surface[ok] - portees[0]) / dr).astype(int)
162     for k in range(-demi_fenetre, demi_fenetre + 1):
163         m = np.clip(n + k, 0, X.shape[1] - 1)
164         v = X[np.arange(len(X)), m]
165         out = np.where(ok, np.nan_to_num(out) + v, np.nan)
166     return out
167
168
169 def ecarts(Z, relief, zone):
170     x0, x1, y0, y1 = zone
171     jx = (relief.x >= x0) & (relief.x <= x1)
172     iy = (relief.y >= y0) & (relief.y <= y1)
173     e = Z[np.ix_(iy, jx)] - relief.z[np.ix_(iy, jx)]
174     ok = np.isfinite(e)
175     v = e[ok]
176     return dict(biais=float(v.mean()), ecart_type=float(v.std()), rms=float(np.sqrt(np.mean(v**2))),
177                couverture=float(ok.mean()), p95=float(np.percentile(np.abs(v), 95)))

```

Annexe 3 : la sphère

```
1 from __future__ import annotations
2
3 from dataclasses import dataclass
4
5 import numpy as np
6
7 AXE_FAISCEAU = np.array([0.0, 0.0, -1.0])
8 G = 9.81
9 ANGLE_MAGIQUE = float(np.arctan(1 / np.sqrt(2)))
10
11
12 def qmul(p, q):
13     pw, px, py, pz = np.moveaxis(np.asarray(p, dtype=float), -1, 0)
14     qw, qx, qy, qz = np.moveaxis(np.asarray(q, dtype=float), -1, 0)
15     return np.stack([pw * qw - px * qx - py * qy - pz * qz,
16                     pw * qx + px * qw + py * qz - pz * qy,
17                     pw * qy - px * qz + py * qw + pz * qx,
18                     pw * qz + px * qy - py * qx + pz * qw], axis=-1)
19
20
21 def qconj(q):
22     q = np.asarray(q, dtype=float)
23     return q * np.array([1.0, -1.0, -1.0, -1.0])
24
25
26 def qnorm(q):
27     q = np.asarray(q, dtype=float)
28     return q / np.linalg.norm(q, axis=-1, keepdims=True)
29
30
31 def qrot(q, v):
32     q = np.asarray(q, dtype=float)
33     v = np.asarray(v, dtype=float)
34     w = q[..., :1]
35     u = q[..., 1:]
36     t = 2 * np.cross(u, v)
37     return v + w * t + np.cross(u, t)
38
39
40 def qexp(r):
41     r = np.asarray(r, dtype=float)
42     a = np.linalg.norm(r, axis=-1, keepdims=True)
43     s = np.where(a > 1e-12, np.sin(a / 2) / np.where(a > 1e-12, a, 1.0), 0.5)
44     return np.concatenate([np.cos(a / 2), s * r], axis=-1)
45
46
47 def qlog(q):
48     q = np.asarray(q, dtype=float)
49     q = np.where(q[..., :1] < 0, -q, q)
50     v = q[..., 1:]
51     s = np.linalg.norm(v, axis=-1, keepdims=True)
52     a = 2 * np.arctan2(s, q[..., :1])
53     return np.where(s > 1e-12, a * v / np.where(s > 1e-12, s, 1.0), 2 * v)
54
55
56 def qslerp(p, q, t):
57     p, q = np.asarray(p, dtype=float), np.asarray(q, dtype=float)
58     if np.dot(p, q) < 0:
59         q = -q
60     return qmul(qexp(np.asarray(t)[..., None] * qlog(qmul(q, qconj(p))))), p)
61
62
63 def qmatrice(q):
64     w, x, y, z = q
65     return np.array([[1 - 2 * (y * y + z * z), 2 * (x * y - w * z), 2 * (x * z + w * y)],
66                     [2 * (x * y + w * z), 1 - 2 * (x * x + z * z), 2 * (y * z - w * x)],
67                     [2 * (x * z - w * y), 2 * (y * z + w * x), 1 - 2 * (x * x + y * y)])
68
69
70 def faisceau(q):
71     return qrot(q, AXE_FAISCEAU)
72
73
74 def basculement(d):
75     d = np.asarray(d, dtype=float)
```

```

76     d = d / np.linalg.norm(d, axis=-1, keepdims=True)
77     e = np.broadcast_to(AXE_FAISCEAU, d.shape)
78     w = 1.0 + np.sum(e * d, axis=-1, keepdims=True)
79     q = np.concatenate([w, np.cross(e, d)], axis=-1)
80     return qnorm(q)
81
82
83 def direction_pq(p, q):
84     p, q = np.asarray(p, dtype=float), np.asarray(q, dtype=float)
85     tau = np.hypot(p, q)
86     phi = np.arctan2(q, p)
87     return np.stack([np.sin(tau) * np.cos(phi), np.sin(tau) * np.sin(phi), -np.cos(tau)], axis=-1)
88
89
90 def pq_direction(d):
91     d = np.asarray(d, dtype=float)
92     tau = np.arccos(np.clip(-d[..., 2], -1, 1))
93     phi = np.arctan2(d[..., 1], d[..., 0])
94     return tau * np.cos(phi), tau * np.sin(phi)
95
96
97 @dataclass
98 class Entrainement:
99     R: float = 0.050
100     r_roue: float = 0.019
101     alpha: float = ANGLE_MAGIQUE
102     azimut0: float = np.pi / 2
103     micropas_tour: int = 3200
104
105     def geometrie(self):
106         psi = self.azimut0 + np.radians([0.0, 120.0, 240.0])
107         sa, ca = np.sin(self.alpha), np.cos(self.alpha)
108         n = np.stack([sa * np.cos(psi), sa * np.sin(psi), np.full(3, ca)], axis=1)
109         t = np.stack([-np.sin(psi), np.cos(psi), np.zeros(3)], axis=1)
110         w = np.cross(n, t)
111         return n, t, w
112
113     @property
114     def W(self):
115         return self.geometrie()[2]
116
117     def vitesses_roues(self, omega):
118         return -(self.R / self.r_roue) * (np.asarray(omega) @ self.W.T)
119
120     def omega(self, phi_point):
121         return -(self.r_roue / self.R) * np.linalg.solve(self.W, np.asarray(phi_point).T).T
122
123     def conditionnement(self) -> float:
124         s = np.linalg.svd(self.W, compute_uv=False)
125         return float(s[0] / s[-1])
126
127     @property
128     def rad_par_micropas(self) -> float:
129         return 2 * np.pi / self.micropas_tour
130
131     def micropas(self, omega):
132         return self.vitesses_roues(omega) / self.rad_par_micropas
133
134     def omega_micropas(self, v_micropas):
135         return self.omega(np.asarray(v_micropas) * self.rad_par_micropas)
136
137
138 def conditionnement_alpha(alpha):
139     return Entrainement(alpha=alpha).conditionnement()
140
141
142 @dataclass
143 class Sphere:
144     R: float = 0.050
145     masse: float = 0.150
146     J: float = 1.6e-4
147     excentrement: float = 0.002
148     frottement: float = 0.4
149     rho_eau: float = 998.2
150
151     @property

```

```

152 def volume(self) -> float:
153     return 4 / 3 * np.pi * self.R**3
154
155 def poussees(self) -> float:
156     return self.rho_eau * G * self.volume
157
158 def masse_neutre(self) -> float:
159     return self.rho_eau * self.volume
160
161 def precharge(self, masse=None):
162     m = self.masse if masse is None else masse
163     return self.poussees() - np.asarray(m) * G
164
165 def effort_normal(self, entrainement: Entrainement, masse=None):
166     return self.precharge(masse) / (3 * np.cos(entrainement.alpha))
167
168 def couple_max(self, entrainement: Entrainement, masse=None):
169     return self.frottement * self.essai_normal(entrainement, masse) * self.R
170
171 def couple_pesanteur(self, q):
172     rG = self.excentrement * faisceau(q)
173     return np.cross(rG, np.array([0.0, 0.0, -self.masse * G]))
174
175 def couple_visqueux(self, omega, nu=1.0e-6, mu=1.0e-3):
176     Om = np.linalg.norm(omega)
177     if Om == 0:
178         return np.zeros(3)
179     delta = np.sqrt(nu / Om)
180     return -8 * np.pi / 3 * mu * self.R**4 / delta * np.asarray(omega)
181
182 def acceleration_max(self, entrainement: Entrainement, tau_max=None):
183     c = self.couple_max(entrainement) if tau_max is None else tau_max
184     return c / self.J
185
186
187 def angles_tourelle(d):
188     d = np.asarray(d, dtype=float)
189     return np.arctan2(d[... , 1], d[... , 0]), np.arccos(np.clip(-d[... , 2], -1, 1))

```

Annexe 4 : mesure et asservissement de l'orientation

```
1 from __future__ import annotations
2
3 from dataclasses import dataclass, field
4
5 import numpy as np
6
7 from .sphere import G, qconj, qexp, qmul, qnorm, qrot, qslerp
8
9
10 @dataclass
11 class CalibrationImu:
12     lsb_gyro: float = 65.5
13     lsb_accel: float = 16384.0
14     biais_accel: np.ndarray = field(default_factory=lambda: np.zeros(3))
15     echelle_accel: np.ndarray = field(default_factory=lambda: np.ones(3))
16
17     def gyro(self, brut):
18         return np.radians(np.asarray(brut, dtype=float) / self.lsb_gyro)
19
20     def accel(self, brut):
21         a = np.asarray(brut, dtype=float) / self.lsb_accel * G
22         return (a - self.biais_accel) / self.echelle_accel
23
24
25 def etalonner_accelerometre(mesures):
26     from scipy.optimize import least_squares
27     A = np.asarray(mesures, dtype=float)
28
29     def residus(x):
30         return np.linalg.norm((A - x[:3]) / x[3:], axis=1) - G
31
32     sol = least_squares(residus, np.r_[np.zeros(3), np.ones(3)])
33     return sol.x[:3], sol.x[3:]
34
35
36 class Estimateur:
37     def __init__(self, kp: float = 0.5, ki: float = 0.02, k_arret: float = 0.05, q0=None):
38         self.kp = kp
39         self.ki = ki
40         self.k_arret = k_arret
41         self.q = np.array([1.0, 0.0, 0.0, 0.0]) if q0 is None else qnorm(np.asarray(q0, float))
42         self.biais = np.zeros(3)
43         self.t = None
44         self.histoire_t: list[float] = []
45         self.histoire_q: list[np.ndarray] = []
46         self.histoire_b: list[np.ndarray] = []
47
48     def initialiser(self, accel_moyen, gyro_moyen=None, lacet: float = 0.0):
49         a = np.asarray(accel_moyen, float)
50         a = a / np.linalg.norm(a)
51         z = np.array([0.0, 0.0, 1.0])
52         q = qnorm(np.r_[1.0 + a @ z, np.cross(a, z)]) if a @ z > -0.999 else np.array([0.0, 1.0, 0.0, 0.0])
53         self.q = qmul(qexp(np.array([0.0, 0.0, lacet])), q)
54         if gyro_moyen is not None:
55             self.biais = np.asarray(gyro_moyen, float).copy()
56
57     def mesure(self, t: float, gyro, accel, immobile: bool = False):
58         gyro = np.asarray(gyro, float)
59         if self.t is None:
60             self.t = t
61             self._noter()
62             return
63         dt = t - self.t
64         if dt <= 0:
65             return
66         self.t = t
67         correction = np.zeros(3)
68         na = np.linalg.norm(accel)
69         if abs(na - G) < 0.1 * G:
70             a = np.asarray(accel) / na
71             v = qrot(qconj(self.q), np.array([0.0, 0.0, 1.0]))
72             e = np.cross(a, v)
73             correction = self.kp * e
74             self.biais = self.biais - self.ki * e * dt
75         if immobile:
```

```

76         self.biais = self.biais + self.k_arret * (gyro - self.biais)
77         w = correction
78     else:
79         w = gyro - self.biais + correction
80         self.q = qnorm(qmul(self.q, qexp(w * dt)))
81         self._noter()
82
83     def _noter(self):
84         self.histoire_t.append(self.t)
85         self.histoire_q.append(self.q.copy())
86         self.histoire_b.append(self.biais.copy())
87
88     def predire(self, t: float, omega_bassin):
89         if self.t is None:
90             return self.q
91         return qnorm(qmul(qexp(np.asarray(omega_bassin) * max(0.0, t - self.t)), self.q))
92
93     def historique(self):
94         return np.array(self.histoire_t), np.array(self.histoire_q), np.array(self.histoire_b)
95
96
97     def interpoler_orientations(t_ref, q_ref, t):
98         t_ref = np.asarray(t_ref)
99         q_ref = np.asarray(q_ref)
100         i = np.clip(np.searchsorted(t_ref, t) - 1, 0, len(t_ref) - 2)
101         out = np.empty((len(t), 4))
102         for k, (ik, tk) in enumerate(zip(i, t)):
103             u = (tk - t_ref[i]) / max(t_ref[i + 1] - t_ref[i], 1e-9)
104             out[k] = qslerp(q_ref[i], q_ref[i + 1], float(np.clip(u, 0, 1)))
105         return out
106
107
108     class Horloge:
109         def __init__(self, fenetre: float = 5.0):
110             self.fenetre = fenetre
111             self.ecarts: list[tuple[float, float]] = []
112             self.decalage = None
113             self._dernier_brut = None
114             self._tours = 0
115
116         def deplier(self, t_us: int) -> float:
117             if self._dernier_brut is not None and t_us < self._dernier_brut - 2**31:
118                 self._tours += 1
119                 self._dernier_brut = t_us
120                 return (t_us + self._tours * 2**32) * 1e-6
121
122         def observer(self, t_appareil: float, t_reception: float):
123             e = t_reception - t_appareil
124             self.ecarts.append((t_reception, e))
125             while self.ecarts and self.ecarts[0][0] < t_reception - self.fenetre:
126                 self.ecarts.pop(0)
127             self.decalage = min(x[1] for x in self.ecarts)
128
129         def vers_hote(self, t_appareil):
130             return np.asarray(t_appareil) + (self.decalage or 0.0)
131
132
133     1 from __future__ import annotations
134     2
135     3 from dataclasses import dataclass
136     4
137     5 import numpy as np
138     6
139     7 from .sphere import basculement, direction_pq, qconj, qlog, qmul
140     8
141     9
142     10 @dataclass
143     11 class Segment:
144     12     t0: float
145     13     duree: float
146     14     debut: np.ndarray
147     15     fin: np.ndarray
148     16     v_max: float
149     17     a_max: float
150     18     immobile: bool = False
151
152     19
153     20     def position(self, t):
154     21         if self.immobile:

```

```

22         return self.debut
23     s_tot = np.linalg.norm(self.fin - self.debut)
24     tau = np.clip(t - self.t0, 0, self.duree)
25     ta = min(self.v_max / self.a_max, self.duree / 2)
26     v = self.a_max * ta
27     if tau < ta:
28         s = 0.5 * self.a_max * tau**2
29     elif tau > self.duree - ta:
30         s = s_tot - 0.5 * self.a_max * (self.duree - tau) ** 2
31     else:
32         s = 0.5 * self.a_max * ta**2 + v * (tau - ta)
33     return self.debut + (self.fin - self.debut) * (s / s_tot if s_tot > 0 else 0)
34
35
36 def duree_trapeze(distance, v_max, a_max):
37     if distance <= v_max**2 / a_max:
38         return 2 * np.sqrt(distance / a_max)
39     return distance / v_max + v_max / a_max
40
41
42 class Balayage:
43     def __init__(self, demi_largeur=np.radians(35), pas_lignes=np.radians(1.0), v_max=0.87, a_max=6.0,
44                 arret=0.15, pause_initiale=2.0):
45         self.segments: list[Segment] = []
46         t = 0.0
47         pos = np.zeros(2)
48
49         def arreter(duree):
50             nonlocal t
51             self.segments.append(Segment(t, duree, pos.copy(), pos.copy(), v_max, a_max, True))
52             t += duree
53
54         def aller(cible):
55             nonlocal t, pos
56             cible = np.asarray(cible, float)
57             d = np.linalg.norm(cible - pos)
58             if d < 1e-12:
59                 return
60             T = duree_trapeze(d, v_max, a_max)
61             self.segments.append(Segment(t, T, pos.copy(), cible, v_max, a_max))
62             t += T
63             pos = cible
64
65         arreter(pause_initiale)
66         L = demi_largeur
67         n = int(round(2 * L / pas_lignes)) + 1
68         for k, qk in enumerate(np.linspace(-L, L, n)):
69             depart, arrivee = (-L, L) if k % 2 == 0 else (L, -L)
70             aller((depart, qk))
71             arreter(arret)
72             aller((arrivee, qk))
73             arreter(arret)
74         aller((0.0, 0.0))
75         arreter(pause_initiale)
76         self.duree = t
77         self._t0 = np.array([s.t0 for s in self.segments])
78
79     def pq(self, t):
80         i = int(np.clip(np.searchsorted(self._t0, t, side="right") - 1, 0, len(self.segments) - 1))
81         s = self.segments[i]
82         return s.position(t), s.immobile
83
84     def consigne(self, t, h=1e-3):
85         (p, q), immobile = self.pq(t)
86         qd = basculement(direction_pq(p, q))
87         (p2, q2), _ = self.pq(t + h)
88         qd2 = basculement(direction_pq(p2, q2))
89         omega = qlog(qmul(qd2, qconj(qd))) / h
90         return qd, omega, immobile
91
92
93 class Regulateur:
94     def __init__(self, k: float = 5.0, omega_max: float = 1.5, a_max: float = 10.0):
95         self.k = k
96         self.omega_max = omega_max
97         self.a_max = a_max

```

```

98     self.precedent = np.zeros(3)
99
100 def commande(self, q_est, q_d, omega_d, dt):
101     qe = qmul(q_d, qconj(q_est))
102     if qe[0] < 0:
103         qe = -qe
104     w = np.asarray(omega_d) + self.k * 2 * qe[1:]
105     n = np.linalg.norm(w)
106     if n > self.omega_max:
107         w *= self.omega_max / n
108     dw = w - self.precedent
109     n = np.linalg.norm(dw)
110     if n > self.a_max * dt:
111         w = self.precedent + dw * (self.a_max * dt / n)
112     self.precedent = w
113     return w

```

Annexe 5 : carte du berceau

```
1  #pragma once
2
3  #include <stdint.h>
4
5  namespace pas {
6
7  struct Moteur {
8      int32_t consigne = 0;
9      int32_t vitesse = 0;
10     uint32_t increment = 0;
11     uint32_t phase = 0;
12     int32_t position = 0;
13     bool sens = true;
14 };
15
16 class Generateur {
17 public:
18     explicit Generateur(uint32_t f_interruption, int32_t acceleration_max)
19         : f_(f_interruption), a_(acceleration_max) {}
20
21     Moteur m[3];
22
23     void rampe(uint32_t dt_us) {
24         int32_t dv_max = static_cast<int32_t>((static_cast<int64_t>(a_) * dt_us) / 1000000);
25         if (dv_max < 1) dv_max = 1;
26         for (auto &x : m) {
27             int32_t dv = x.consigne - x.vitesse;
28             if (dv > dv_max) dv = dv_max;
29             if (dv < -dv_max) dv = -dv_max;
30             x.vitesse += dv;
31             uint32_t v = static_cast<uint32_t>(x.vitesse < 0 ? -x.vitesse : x.vitesse);
32             x.increment = static_cast<uint32_t>((static_cast<uint64_t>(v) << 32) / f_);
33             x.sens = x.vitesse >= 0;
34         }
35     }
36
37     uint8_t tic() {
38         uint8_t masque = 0;
39         for (int i = 0; i < 3; i++) {
40             Moteur &x = m[i];
41             uint32_t avant = x.phase;
42             x.phase += x.increment;
43             if (x.phase < avant) {
44                 x.position += x.sens ? 1 : -1;
45                 masque |= static_cast<uint8_t>(1u << i);
46             }
47         }
48         return masque;
49     }
50
51     void arret_immediat() {
52         for (auto &x : m) { x.consigne = 0; x.vitesse = 0; x.increment = 0; }
53     }
54
55     bool immobile() const {
56         return m[0].vitesse == 0 && m[1].vitesse == 0 && m[2].vitesse == 0;
57     }
58
59 private:
60     uint32_t f_;
61     int32_t a_;
62 };
63
64 }
```

```
1  #include <stdio.h>
2  #include <string.h>
3
4  #include "hardware/gpio.h"
5  #include "hardware/irq.h"
6  #include "hardware/sync.h"
7  #include "hardware/uart.h"
8  #include "pico/stdio_usb.h"
9  #include "pico/stdlib.h"
10 #include "pas.h"
```

```

11 #include "protocole.h"
12
13 constexpr uint PIN_STEP[3] = {2, 4, 6};
14 constexpr uint PIN_DIR[3] = {3, 5, 7};
15 constexpr uint PIN_EN = 10;
16 constexpr uint PIN_OPTIQUE_TX = 20;
17 constexpr uint PIN_OPTIQUE_RX = 21;
18 constexpr uint32_t DEBIT_OPTIQUE = 2000000;
19 constexpr uint32_t F_TIC = 50000;
20 constexpr int32_t ACCELERATION = 40000;
21 constexpr int32_t V_MAX = 20000;
22 constexpr uint32_t CHIEN_US = 200000;
23 constexpr uint32_t PERIODE_RAMPE_US = 1000;
24 constexpr uint32_t PERIODE_ODO_US = 10000;
25
26 #define UART_OPTIQUE uart1
27
28 static pas::Générateur generateur(F_TIC, ACCELERATION);
29 static uint32_t masque_step = 0;
30
31 constexpr uint32_t TAILLE_ANNEAU = 8192;
32 static volatile uint8_t anneau[TAILLE_ANNEAU];
33 static volatile uint32_t tete = 0;
34 static uint32_t queue_anneau = 0;
35
36 static void reception_optique() {
37     while (uart_is_readable(UART_OPTIQUE)) {
38         anneau[tete & (TAILLE_ANNEAU - 1)] = static_cast<uint8_t>(uart_getc(UART_OPTIQUE));
39         tete = tete + 1;
40     }
41 }
42
43 static bool tic(repeating_timer_t *) {
44     gpio_clr_mask(masque_step);
45     uint8_t m = generateur.tic();
46     uint32_t sortie = 0;
47     for (int i = 0; i < 3; i++)
48         if (m & (1u << i)) sortie |= 1u << PIN_STEP[i];
49     gpio_set_mask(sortie);
50     return true;
51 }
52
53 static uint8_t trame[proto::TRAME_MAX];
54
55 static void vers_ordinateur(uint8_t type, const void *charge, uint16_t n) {
56     size_t t = proto::encoder(type, charge, n, trame);
57     fwrite(trame, 1, t, stdout);
58 }
59
60 static void envoyer_odometrie(bool chien, bool actif) {
61     proto::Odometrie o{};
62     uint32_t s = save_and_disable_interrupts();
63     o.t_us = time_us_32();
64     for (int i = 0; i < 3; i++) {
65         o.position[i] = generateur.m[i].position;
66         o.vitesse[i] = generateur.m[i].vitesse;
67     }
68     restore_interrupts(s);
69     o.drapeaux = (chien ? 1 : 0) | (actif ? 2 : 0);
70     vers_ordinateur(proto::ODOMETRIE, &o, sizeof o);
71 }
72
73 int main() {
74     stdio_init_all();
75     stdio_set_translate_crlf(&stdio_usb, false);
76     for (int i = 0; i < 3; i++) {
77         gpio_init(PIN_STEP[i]);
78         gpio_set_dir(PIN_STEP[i], GPIO_OUT);
79         gpio_init(PIN_DIR[i]);
80         gpio_set_dir(PIN_DIR[i], GPIO_OUT);
81         masque_step |= 1u << PIN_STEP[i];
82     }
83     gpio_init(PIN_EN);
84     gpio_set_dir(PIN_EN, GPIO_OUT);
85     gpio_put(PIN_EN, 1);
86

```

```

87  uart_init(UART_OPTIQUE, DEBIT_OPTIQUE);
88  gpio_set_function(PIN_OPTIQUE_TX, GPIO_FUNC_UART);
89  gpio_set_function(PIN_OPTIQUE_RX, GPIO_FUNC_UART);
90  irq_set_exclusive_handler(UART1_IRQ, reception_optique);
91  irq_set_enabled(UART1_IRQ, true);
92  uart_set_irq_enables(UART_OPTIQUE, true, false);
93
94  repeating_timer_t minuterie;
95  add_repeating_timer_us(-static_cast<int64_t>(1000000 / F_TIC), tic, nullptr, &minuterie);
96
97  proto::Decodeur depuis_ordinateur;
98  proto::Decodeur depuis_sonde;
99  uint32_t derniere_consigne = time_us_32();
100  uint32_t prochaine_rampe = derniere_consigne;
101  uint32_t prochaine_odo = derniere_consigne;
102  bool chien = true, actif = false;
103
104  while (true) {
105      int c;
106      while ((c = getchar_timeout_us(0)) != PICO_ERROR_TIMEOUT) {
107          if (!depuis_ordinateur.ajouter(static_cast<uint8_t>(c))) continue;
108          if (depuis_ordinateur.type == proto::CONSIGNE_ROUES && depuis_ordinateur.n == sizeof(proto::ConsigneRoues))
↪ {
109              proto::ConsigneRoues cr;
110              memcpy(&cr, depuis_ordinateur.charge, sizeof cr);
111              for (int i = 0; i < 3; i++) {
112                  int32_t v = cr.v[i];
113                  generateur.m[i].consigne = v > V_MAX ? V_MAX : (v < -V_MAX ? -V_MAX : v);
114              }
115              actif = cr.actif != 0;
116              gpio_put(PIN_EN, actif ? 0 : 1);
117              derniere_consigne = time_us_32();
118              chien = false;
119          } else if (depuis_ordinateur.type == proto::DEMARRER || depuis_ordinateur.type == proto::ARRETER) {
120              size_t t = proto::encoder(depuis_ordinateur.type, depuis_ordinateur.charge, depuis_ordinateur.n, trame);
121              uart_write_blocking(UART_OPTIQUE, trame, t);
122          }
123      }
124
125      while (queue_anneau != tete) {
126          uint8_t o = anneau[queue_anneau & (TAILLE_ANNEAU - 1)];
127          queue_anneau++;
128          if (depuis_sonde.ajouter(o)) vers_ordinateur(depuis_sonde.type, depuis_sonde.charge, depuis_sonde.n);
129      }
130
131      uint32_t t = time_us_32();
132      if (t - derniere_consigne > CHIEN_US) {
133          for (auto &m : generateur.m) m.consigne = 0;
134          chien = true;
135      }
136      if (static_cast<int32_t>(t - prochaine_rampe) >= 0) {
137          prochaine_rampe += PERIODE_RAMPE_US;
138          uint32_t s = save_and_disable_interrupts();
139          generateur.rampe(PERIODE_RAMPE_US);
140          for (int i = 0; i < 3; i++) gpio_put(PIN_DIR[i], generateur.m[i].sens);
141          restore_interrupts(s);
142      }
143      if (static_cast<int32_t>(t - prochaine_odo) >= 0) {
144          prochaine_odo += PERIODE_ODO_US;
145          envoyer_odometrie(chien, actif);
146      }
147      fflush(stdout);
148  }
149  }

```

Annexe 6 : sonde

```
1 .program impulsions
2 .wrap_target
3     pull block
4     mov x, osr
5 periode:
6     set pins, 1 [3]
7     set pins, 2 [2]
8     jmp x-- periode
9     set pins, 0
10 .wrap
11
12 % c-sdk {
13 #include "hardware/clocks.h"
14
15 static inline void impulsions_program_init(PIO pio, uint sm, uint offset, uint pin, float f_machine) {
16     pio_sm_config c = impulsions_program_get_default_config(offset);
17     sm_config_set_set_pins(&c, pin, 2);
18     pio_gpio_init(pio, pin);
19     pio_gpio_init(pio, pin + 1);
20     pio_sm_set_consecutive_pindirs(pio, sm, pin, 2, true);
21     sm_config_set_clkdiv(&c, (float)clock_get_hz(clk_sys) / f_machine);
22     pio_sm_init(pio, sm, offset, &c);
23     pio_sm_set_enabled(pio, sm, true);
24 }
25 %}
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```

50     imu_ecrire(0x1C, 0x00);
51 }
52
53 static bool imu_lire(int16_t gyro[3], int16_t accel[3]) {
54     uint8_t registre = 0x3B, b[14];
55     if (i2c_write_blocking(i2c0, ADRESSE_IMU, &registre, 1, true) != 1) return false;
56     if (i2c_read_blocking(i2c0, ADRESSE_IMU, b, 14, false) != 14) return false;
57     for (int i = 0; i < 3; i++) {
58         accel[i] = static_cast<int16_t>((b[2 * i] << 8) | b[2 * i + 1]);
59         gyro[i] = static_cast<int16_t>((b[8 + 2 * i] << 8) | b[9 + 2 * i]);
60     }
61     return true;
62 }
63
64 static void envoyer(uint8_t type, const void *donnees, uint16_t n) {
65     dma_channel_wait_for_finish_blocking(dma_emission);
66     size_t t = proto::encoder(type, donnees, n, emission);
67     dma_channel_transfer_from_buffer_now(dma_emission, emission, t);
68 }
69
70 static proto::Decodeur ordres;
71
72 static void lire_ordres() {
73     while (uart_is_readable(UART_OPTIQUE)) {
74         if (!ordres.ajouter(static_cast<uint8_t>(uart_getc(UART_OPTIQUE)))) continue;
75         if (ordres.type == proto::DEMARRER && ordres.n == sizeof(proto::Demarrer)) {
76             proto::Demarrer d;
77             memcpy(&d, ordres.charge, sizeof d);
78             cadence = d.cadence_hz ? d.cadence_hz : 50;
79             n_echantillons = d.n > proto::N_MAX ? proto::N_MAX : d.n;
80             cycles = d.cycles ? d.cycles : 1;
81             en_marche = true;
82         } else if (ordres.type == proto::ARRETER) {
83             en_marche = false;
84         }
85     }
86 }
87
88 static void preparer() {
89     uart_init(UART_OPTIQUE, DEBIT_OPTIQUE);
90     gpio_set_function(PIN_OPTIQUE_TX, GPIO_FUNC_UART);
91     gpio_set_function(PIN_OPTIQUE_RX, GPIO_FUNC_UART);
92     dma_emission = dma_claim_unused_channel(true);
93     dma_channel_config e = dma_channel_get_default_config(dma_emission);
94     channel_config_set_transfer_data_size(&e, DMA_SIZE_8);
95     channel_config_set_read_increment(&e, true);
96     channel_config_set_write_increment(&e, false);
97     channel_config_set_dreq(&e, uart_get_dreq(UART_OPTIQUE, true));
98     dma_channel_configure(dma_emission, &e, &uart_get_hw(UART_OPTIQUE)->dr, emission, 0, false);
99
100     adc_init();
101     adc_gpio_init(PIN_ENVELOPPE);
102     adc_select_input(0);
103     adc_fifo_setup(true, true, 1, false, false);
104     adc_set_clkdiv(0);
105     dma_adc = dma_claim_unused_channel(true);
106     dma_channel_config c = dma_channel_get_default_config(dma_adc);
107     channel_config_set_transfer_data_size(&c, DMA_SIZE_16);
108     channel_config_set_read_increment(&c, false);
109     channel_config_set_write_increment(&c, true);
110     channel_config_set_dreq(&c, DREQ_ADC);
111     dma_channel_configure(dma_adc, &c, echantillons, &adc_hw->fifo, 0, false);
112
113     uint offset = pio_add_program(pio, &impulsion_program);
114     sm = pio_claim_unused_sm(pio, true);
115     impulsion_program_init(pio, sm, offset, PIN_IMPULSION, 8.0e6f);
116 }
117
118 static uint32_t tirer(uint16_t n) {
119     adc_run(false);
120     adc_fifo_drain();
121     dma_channel_set_write_addr(dma_adc, echantillons, false);
122     dma_channel_set_trans_count(dma_adc, n, true);
123     uint32_t s = save_and_disable_interrupts();
124     uint32_t t = time_us_32();
125     adc_run(true);

```

```

126     pio_sm_put(pio, sm, static_cast<uint32_t>(cycles - 1));
127     restore_interrupts(s);
128     dma_channel_wait_for_finish_blocking(dma_adc);
129     adc_run(false);
130     adc_fifo_drain();
131     return t;
132 }
133
134 int main() {
135     i2c_init(i2c0, 400000);
136     gpio_set_function(PIN_SDA, GPIO_FUNC_I2C);
137     gpio_set_function(PIN_SCL, GPIO_FUNC_I2C);
138     gpio_pull_up(PIN_SDA);
139     gpio_pull_up(PIN_SCL);
140     imu_initialiser();
141     preparer();
142
143     uint32_t prochaine_imu = time_us_32();
144     uint32_t prochain_ping = prochaine_imu;
145     uint16_t seq_imu = 0;
146     uint32_t seq_ping = 0;
147
148     while (true) {
149         lire_ordres();
150         uint32_t t = time_us_32();
151         if (static_cast<int32_t>(t - prochaine_imu) >= 0) {
152             prochaine_imu += PERIODE_IMU_US;
153             proto::MesureImu m{};
154             if (imu_lire(m.gyro, m.accel)) {
155                 m.t_us = time_us_32();
156                 m.seq = ++seq_imu;
157                 envoyer(proto::IMU, &m, sizeof m);
158             }
159         }
160         if (!en_marche) {
161             prochain_ping = t;
162         } else if (static_cast<int32_t>(t - prochain_ping) >= 0) {
163             prochain_ping += 1000000u / cadence;
164             proto::EnteteSonar e{};
165             e.seq = ++seq_ping;
166             e.n = n_echantillons;
167             e.t_us = tirer(e.n);
168             memcpy(charge, &e, sizeof e);
169             memcpy(charge + sizeof e, echantillons, 2u * e.n);
170             envoyer(proto::PING, charge, static_cast<uint16_t>(sizeof e + 2u * e.n));
171         }
172     }
173 }

```