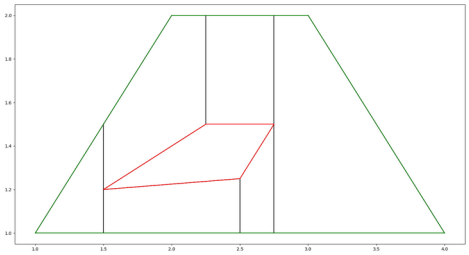
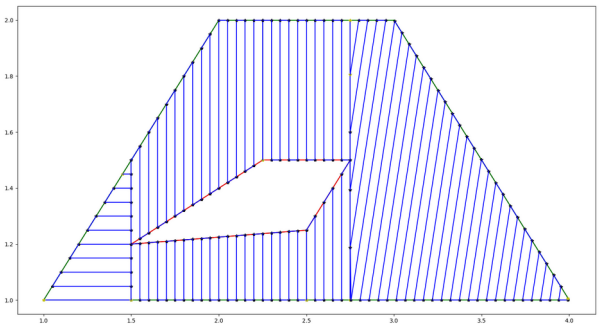
Rapport Architecture Robotique

Kévin AFFRAIX

La séance d'aujourd'hui a été consacrée au test et au de-bug de l'algorithme de décomposition en parties convexes, ainsi que d'un cas particulier du calcul de direction optimale d'un boustrophédon. Après fusion avec le premier algorithme, qui donnait la direction optimale du boustrophédon sur une partie convexe, on obtient :





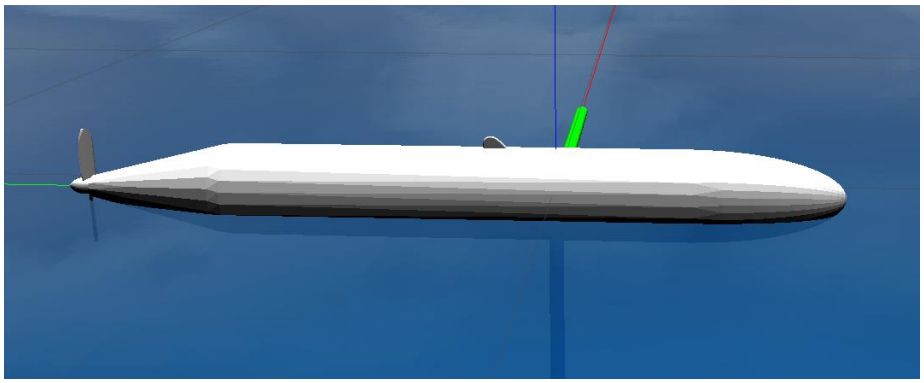
où les points jaunes correspondent au départ et à l'arrivée de chaque boustrophédon.

Le travail de la prochaine séance sera de déterminer dans quel ordre parcourir les boustrophédons (classe les points jaunes) pour minimiser la distance parcourue non utile. Il faudrait aussi implémenter la fusoin des parties convexes adjacentes de même direction de parcours.

Compte-rendu Riptide (14/12/2020)

Simulation du Riptide : **Romane FLECHARD, Mourtaza KASSAMALY et Quentin VINTRAS**

- **En cours :**
  - Finir la modélisation du Riptide (positions relatives et commande du propulseur et des ailerons) ;



  - Il reste à gérer la position des ailerons et de gérer les topics ROS associés aux actionneurs.
- **A faire :**
  - Trouver tous les noms des topics auxquels on va s'abonner (topics des capteurs) et sur lesquels on va publier les commandes (topics des actionneurs) ;
  - Pour chaque topic, déterminer le format du message qui lui est associé ;
  - Sous rqt, s'entraîner à récupérer les données capteurs et à envoyer des commandes aux actionneurs du Riptide ;
  - Faire les codes ROS qui publient des commandes dans les topics actionneurs et souscrit aux topics capteurs ;
  - Lier ces codes au contrôleur codé par Julien et Paul (récupérer la commande des actionneurs et la transformer en commande pour la simulation).

Contrôleur du Riptide : **Julien PIRANDA et Paul PINEAU**

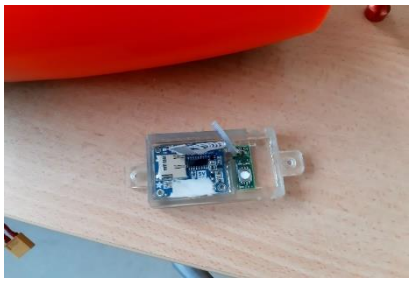
Aujourd'hui, nous avons continué le passage sous ROS du riptide. Nous nous sommes occupés de la partie mission du projet. Avec la mise en place de waypoints, de lecture de fichier Json.

Le nœud Ros Mission permet d'envoyer les waypoints sous la forme de geometry_msgs/Points. De plus, nous nous sommes intéressés au traitement des données reçues par les capteurs.

Prochaine étape : mise en place d'un filtre de kalman.

Electronique du Riptide :

- **Hamid HACENE**
  - **Fait :**
    - Changement du capteur de pression (illustration ci-dessous) :
      - Détachement de l'ancien capteur ;
      - Fraisage d'un trou pour le nouveau capteur dans la coque de l'AUV ;
      - Insertion du capteur bar30 de Blue Robotics ;
      - Test de l'étanchéité du capteur.
  - **A faire :**
    - Test du capteur de pression ;
    - Ecriture des driver ROS pour le capteur et test en piscine.



- **Corentin LEMOINE**
  - **Fait :**
    - Implémentation du code permettant la reprise à distance de l'AUV via la télécommande ;
    - Définition du comportement lors de la perte de signal de la commande.

$$u = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} 0 & \frac{-1}{\sqrt{3}} & -1 \\ \frac{-1}{2} & \frac{\sqrt{3}}{2} & \frac{1}{2} \\ \frac{-1}{2} & \frac{-\sqrt{3}}{6} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$$

```
  cmdRoll = pwm2cmd(pwmRoll);
  cmdPitch = pwm2cmd(pwmPitch);
  cmdYaw = pwm2cmd(pwmYaw);
  cmdThrottle = pwm2cmd(pwmThrottle);

  u1 = -0.5773502691896258 * cmdPitch - cmdYaw + cmdThrottle;
//min : -2,577... max : 2,577...
  u2 = -0.5 * cmdRoll + 0.8660254037844386 * cmdPitch + 0.5 * cmdYaw + cmdThrottle;
//-2.8660254037844384
  u3 = -0.5 * cmdRoll - 0.28867513459481287 * cmdPitch + 0.5 * cmdYaw + cmdThrottle;
//-2.28867513459481287

  float normalizedU1 = map(u1, -2.5773502691896258, 2.5773502691896258, 0, 1);
  float normalizedU2 = map(u2, -2.8660254037844384, 2.8660254037844384, 0, 1);
  float normalizedU3 = map(u3, -2.28867513459481287, 2.28867513459481287, 0, 1);

  servos[0].writeMicroseconds(cmd2pwm(normalizedU1));
  servos[1].writeMicroseconds(cmd2pwm(normalizedU2));
  servos[2].writeMicroseconds(cmd2pwm(normalizedU3));
  servos[3].writeMicroseconds(pwmSwitch);
```

  - **En cours (à faire) :**
    - Développement de l'interface de création de mission.

Groupe magnétomètre : rapport du 14 décembre

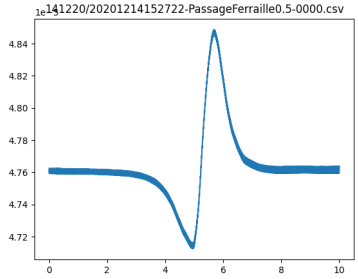
Agathe, Alexandre, Robin :

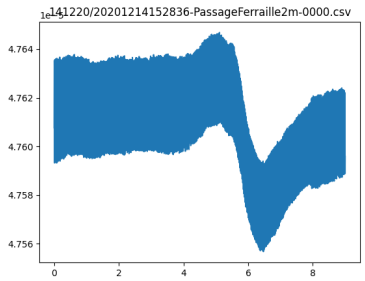
1. Tracé et analyse des données de logs du magnétomètre acquises la semaine précédente. Du bruit apparaît sur les graphes, nous décidons de reproduire l'expérience avec des scénarios plus spécifiques pour valider l'hypothèse de détection.

2. Discussion avec Romain Schwab pour utiliser un câble plus long (des raccords de connectique devront être faits).

3. Expérience sur le terrain :
- scénario pour acquérir le bruit ambiant
- scénario passage à vide (magnéto statique)
- scénario passage ferraille 0.5/1/2 m (magnéto statique)
- scénario passage pot en fonte 1 m (magnéto statique)
-scénarios similaires avec magnétomètre dynamique (obstacles statiques)

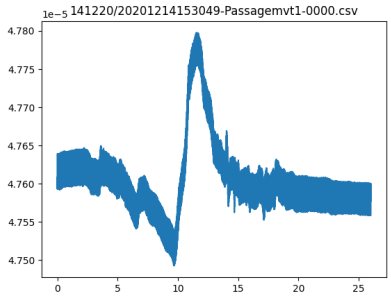
4. Tracé et analyse des données de logs du magnétomètre. Ré-échantillonnage du timestamp qui était mauvais. Voici quelques figures obtenues (norme globale du champ magnétique en mT) :

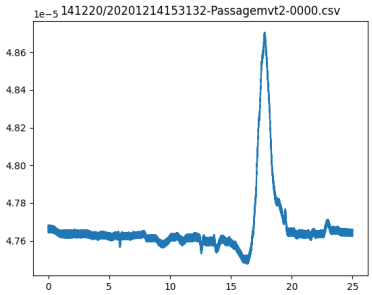


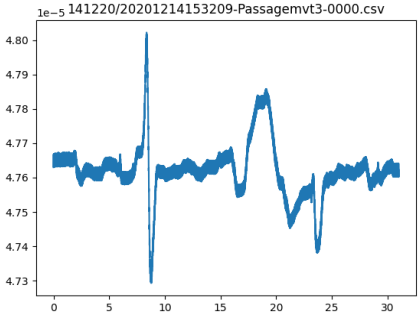


Scénario passage de la ferraille à 0.5m Scénario passage de la ferraille à 2m

On constate une relation entre le bruit des mesures et la distance séparant le magnétomètre de la plaque de ferraille : plus l'objet est proche, plus la détection magnétique est fine.







Scénarios similaires avec le magnétomètre dynamique

Ici un bruit dû au déplacement du magnétomètre s'ajoute aux mesures. Néanmoins, les perturbations magnétiques sont assez claires pour identifier le passage juste à côté des objets d'intérêt (la dernière traduit un passage près de la plaque de ferraille, puis du pot en fonte)

Magnetic mapping using a trailed magnetometer

Quentin Brateau, Gwendal Priser, Paul-Antoine Le Tolguenec, Jules Berhault[1,✉]

[1]ENSTA Bretagne, Brest

The realization of a magnetic map of a terrain is a powerful tool especially used in mine warfare. This cartography can be done with a magnetometer, but the task is quite slow. This is why it is useful to use robots to make this cartography. The problem induced by this solution is the addition of magnetic disturbances related to the structure of the robot and its actuators. It is then possible to drag the magnetometer on a sled behind the robot.

Magnetic | Mapping | Control | Interval Analysis
Correspondence: ***quentin.brateau@ensta-bretagne.org***

Formalism

A. Mathematical context. The system evolve on a field which should be mapped with the magnetometer. It follows the evolution describe by EQUATION 1.

$$\begin{cases} \dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t)) & (evolution\ equation) \\ \mathbf{y}^i = g(\mathbf{x}(t_i)) & (observation\ equation) \end{cases} \tag{1}$$

where $\mathbf{x}$ is the state of the robot, $\mathbf{u}$ is the input of the system. As $\mathbf{x}$ and $\mathbf{u}$ continuously evolve with time, we define the trajectories of this system denoted by $\mathbf{x}(\cdot)$ and $\mathbf{u}(\cdot)$. We could also define tubes enclosing these trajectories at every time denoted by $\forall t, [\mathbf{x}](t)$ and $\forall t, [\mathbf{u}](t)$. Measurements are available at each t_i and are denoted by $\mathbf{y}^i$. Therefore it is assumed that $\forall t, \mathbf{u}(t) \in [\mathbf{u}](t)$ and that for each measurements $\mathbf{y}^i \in [\mathbf{y}^i]$. This last conditions means that each measurement is bounded which is an assumption made frequently when we are dealing with constrain programming.

B. State equations. The system is composed of a vehicle that will drag a sledge that transports a magnetometer. The towing vehicle is a tank type vehicle, and the sled is attached to it with a rope. Assuming that the rope is constantly under tension, we are able to find the equations describing the dynamics of the system.

$$f(\mathbf{x}(t), \mathbf{u}(t)) = \begin{cases} \dot{\mathbf{x}}_1 = \frac{\mathbf{u}_1 + \mathbf{u}_2}{2}.cos(\mathbf{x}_3) & \textbf{(2a)} \\ \dot{\mathbf{x}}_2 = \frac{\mathbf{u}_1 + \mathbf{u}_2}{2}.sin(\mathbf{x}_3) & \textbf{(2b)} \\ \dot{\mathbf{x}}_3 = \mathbf{u}_2 - \mathbf{u}_1 & \textbf{(2c)} \\ \dot{\mathbf{x}}_4 = -\frac{\mathbf{u}_1 + \mathbf{u}_2}{2}.sin(\mathbf{x}_4) - \dot{\mathbf{x}}_3 & \textbf{(2d)} \end{cases}$$

Here $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ are respectively the abscissa, the ordinate and the heading of the trailing robot, $\mathbf{x}_4$ is the angle of the sled to the tractor vehicle, as shown in FIGURE 1. We can also notice that $\frac{\mathbf{u}_1 + \mathbf{u}_2}{2}$ is related to the speed control of the char robot and that $\mathbf{u}_2 - \mathbf{u}_1$ is the angular acceleration control.

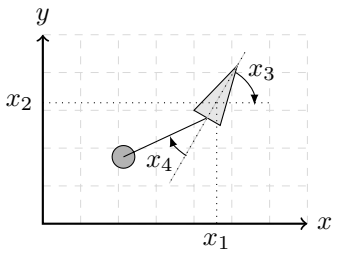


Fig. 1. Diagram representing system state variables

Constrain Network

C. Decomposition. These equations could be broken down into the following set of elementary constrains :

$$\begin{cases} (i) & \mathbf{v}(\cdot) = f(\mathbf{x}(\cdot), \mathbf{u}(\cdot)) \\ (ii) & \dot{\mathbf{x}}(\cdot) = \mathbf{v}(\cdot) \\ (iii) & \dot{\mathbf{v}}(\cdot) = \mathbf{a}(\cdot) \\ (iv) & \mathbf{d}(\cdot) = \mathbf{x}_{1,2}(\cdot) - \mathbf{x_m}(\cdot) \\ (v) & \mathbf{t} = \mathbf{x}_3 + \mathbf{x}_4 \\ (vi) & \mathcal{L}_{polar}\left(\mathbf{d}(\cdot),\ [L],\ \mathbf{t}\right) \\ (vii) & \mathcal{L}_{eval}\left(t_i,\ [\mathbf{y}_{1,2}],\ \mathbf{x}_{1,2}(t_i),\ \mathbf{v}_{1,2}(t_i)\right) \\ (viii) & \mathcal{L}_{eval}\left(t_j,\ [\mathbf{y}_{4,5}],\ \mathbf{v}_{1,2}(t_j),\ \mathbf{a}_{1,2}(t_j)\right) \end{cases} \tag{3}$$

These constrains involve intermediate variables introduced for ease of decomposition: tubes $\mathbf{v}(\cdot) \in \mathbb{R} \to \mathbb{R}^4$ and $\mathbf{a}(\cdot) \in \mathbb{R} \to \mathbb{R}^2$ which are the speed and the acceleration tubes, $\mathbf{d}(\cdot) \in \mathbb{R} \to \mathbb{R}^2$ which is the distance between the sled and the robot, $\mathbf{t}(\cdot) \in \mathbb{R}$ which is the sled angle in the map frame.

Simulator

A python simulation was realized using VIBES in order to test the behavior of the system. A class *Tank* has been created to instantiate a vehicle with its sled. Then the script integrates the evolution equation using Euler's method in order to obtain the state of the system x according to the u inputs. We could see that the behavior of the system seems correct and the model is faithful to reality. Moreover, the GNSS sensor and an accelerometer are simulated in order to enclose the real position in a box.

Reliable set of sled's angle

Considering that the rope remains continuously under tension while the robot is moving, then the evolution of the angle x_4

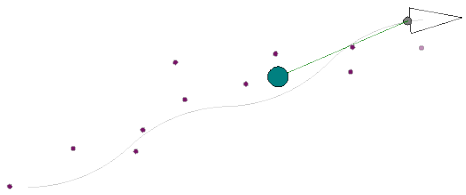


Fig. 2. Simulation of the system

of the sled relative to the towing vehicle is described by the *Differential Equation* **??**.
We are able to find the trajectory of the sled by computing the interval containing the angle x_4, considering that initially x_4 belongs to $[-\pi/2;\pi/2]$. Then by applying the *Differential Equation* **??** on this interval, knowing the control vector u, we end up obtaining a fine interval framing the real angle x_4, independently of the initial angle as we can see on the FIGURE 3.

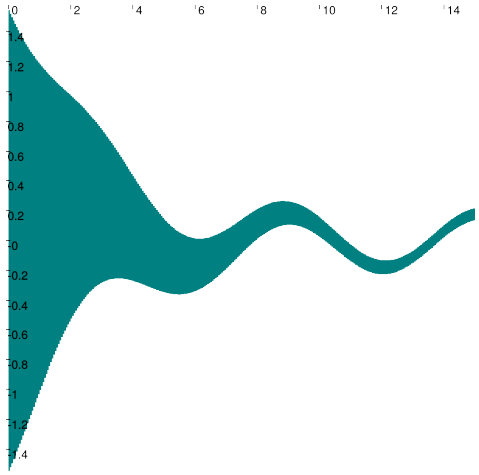


Fig. 3. Integration example using intervals

Sled localization

By having a box enclosing the trailing robot, and an interval framing the angle x_4, we are able to obtain a box containing the sled, knowing the length of the rope.
This box has the shape of a pie sector. *Equation* 4 presents the position of the magnetometer as a function of the state of the system x_1, x_2 and x_4. Here all these variables are intervals. By applying a polar contractor to these intervals, we are able to obtain the intervals containing x_m and y_m.

$$\begin{cases} x_m = x_1 - L.cos(x_4) \\ y_m = x_2 - L.sin(x_4) \end{cases} \quad \textbf{(4)}$$

Magnetometer range of measurement

By knowing a box containing the magnetometer and its measuring range, we are able to find all the points seen for sure and all the points that could be seen by the magnetometer. This allows us to recover the coverage of the area by the magnetometer.

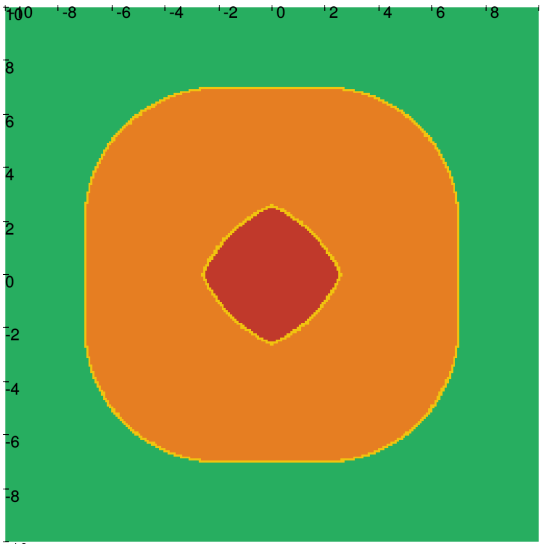


Fig. 4. Measurement range of the magnetometer using Thicksets

The FIGURE 4 shows us these two sets. The green set is the unseen area, the orange set is the area maybe seen and the red set is the area seen for sure by the magnetometer. The yellow set forms the uncertain boundary between the other sets, which can be adjusted. Thus we have a way to know precisely the coverage of the mapping during the mission.

Magnetometer Mapping

With a moving box enclosing the magnetometer at any time, this method allows us to recover the coverage of the area by the magnetometer. Thus we have a way to know precisely the coverage of the mapping during the mission, which depends on the uncertainty on the position of the magnetometer and on the maximum distance of each point of the field to be measured. An example of the covered area of a magnetometer on a field is presented by the FIGURE 5.

Application example

In this part we will see on a practical case what we are able to do with this implementation. We place ourselves in the same conditions as before, i.e. we are in the case of a magnetometer dragged by a robot with tank type evolution equations and the rope must remain tight all along the mission.
Here we want our robot to follow the following Lissajous trajectory:

$$\forall t \in \mathbb{R}, \qquad \mathcal{T}(t) = \begin{cases} x(t) = 20 \times sin\left(\dfrac{2 \times t}{20}\right) & \textbf{(5)} \\ y(t) = 20 \times sin\left(\dfrac{3 \times t}{20}\right) & \textbf{(6)} \end{cases}$$

Fig. 5. Coverage of the map by the magnetometer using Thicksets

$$\forall t \in \mathbb{R}, \quad \frac{d\mathcal{T}(t)}{dt} = \begin{cases} \dfrac{dx(t)}{dt} = 2 \times cos\left(\dfrac{2 \times t}{20}\right) & \textbf{(7)} \\ \dfrac{dy(t)}{dt} = 3 \times cos\left(\dfrac{3 \times t}{20}\right) & \textbf{(8)} \end{cases}$$

$$\forall t \in \mathbb{R}, \quad \frac{d^2\mathcal{T}(t)}{dt^2} = \begin{cases} \dfrac{d^2x(t)}{dt^2} = -\dfrac{4}{20} \times sin\left(\dfrac{2 \times t}{20}\right) & \textbf{(9)} \\ \dfrac{d^2y(t)}{dt^2} = -\dfrac{9}{20} \times sin\left(\dfrac{3 \times t}{20}\right) & \textbf{(10)} \end{cases}$$

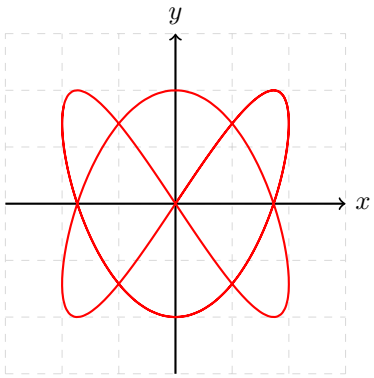


Fig. 6. Lissajous trajectory

The *control.py* file is able to generate the controls to drive the trailing robot. These commands are generated by feedback linearization by controlling the robot in speed and heading. These controls leads to the trajectory in the simulator as we could see on the FIGURE 7

Then the module *mapping.py* allows to process the tubes including the trajectory of the trailing robot and the sledge. The

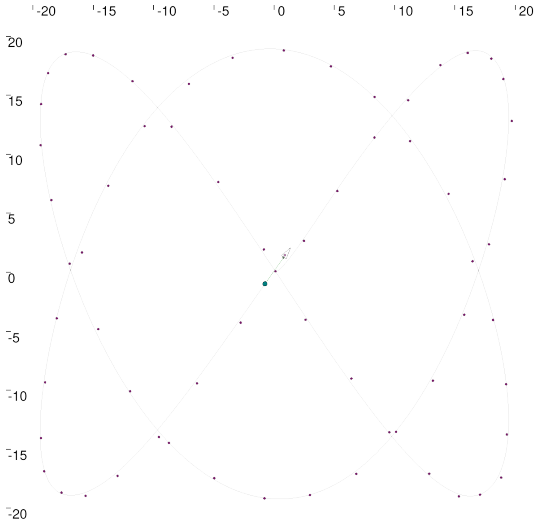


Fig. 7. Simulator's trajectory of the system

tubes are then displayed in order to check that they seem to stick to the trajectory of the system.

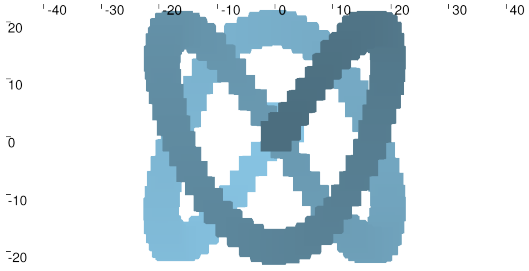


Fig. 8. Trailing vehicle enclosing tube

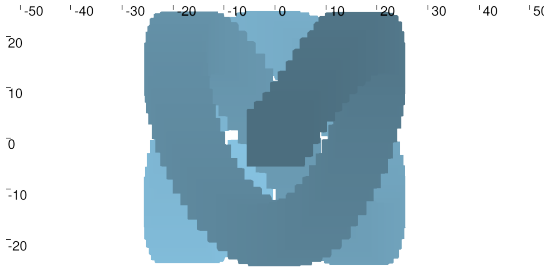


Fig. 9. Magnetometer enclosing tube

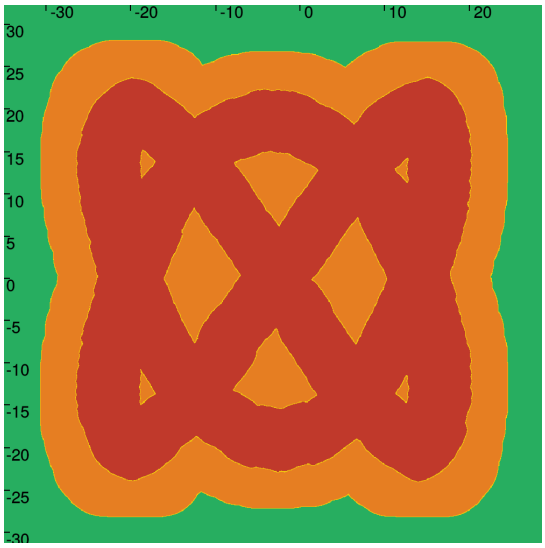


Fig. 10. Magnetometer enclosing tube

Rapport de la séance du 14/12/2020

Mamadou DEMBELE, Maha Halimi ,Florian HÖNISCH

Nous avons effectué la configuration du GPS de Saturne pour qu'il puisse recevoir les corrections RTK. Ensuite nous avons adapté les codes du serveur pour que les corrections RTK soient envoyés vers deux GPS mobiles. C'est-à-dire que le serveur accepte deux connections TCP.

Prochaine séance : On prévoit d'enregistrer des trajectoires avec la luge.

Rapport Simulation Saturne

Aujourd'hui, reprise de la partie de génération du terrain dans lequel le robot évolue.

- Travaille sur l'échelle du terrain et la couleur

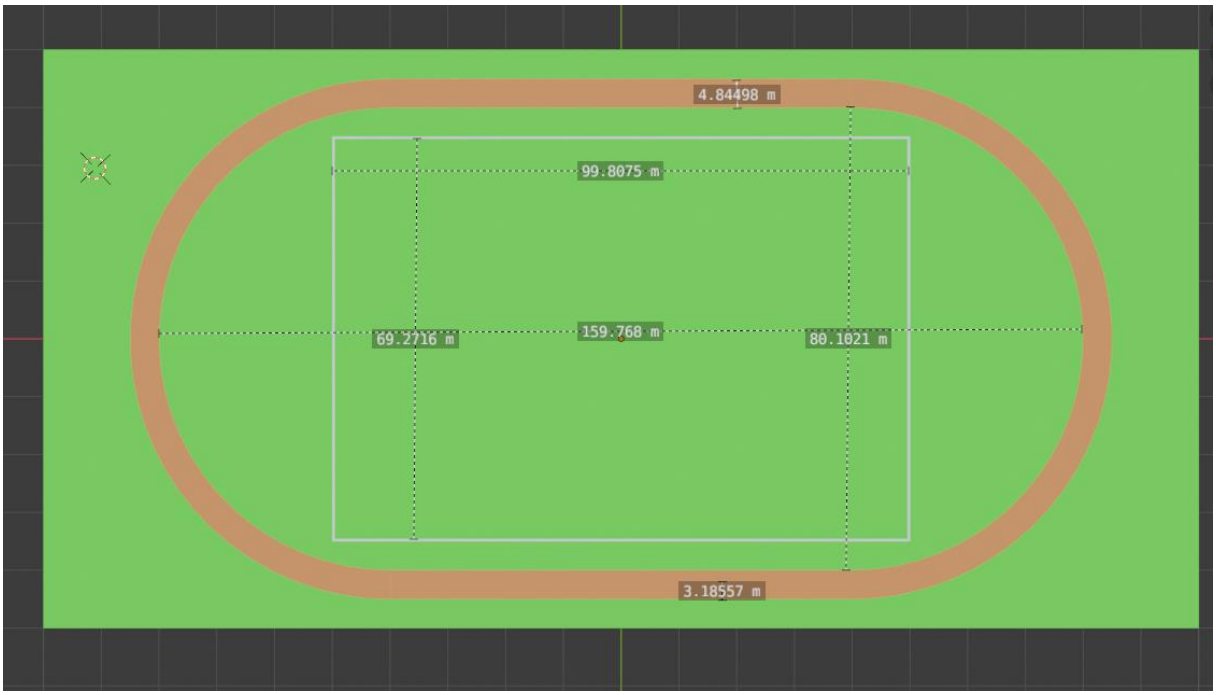


Figure 1 : Le Terrain sous Blender

- Ajout de l'angle du terrain : le terrain n'est pas orienté plein nord

Figure 2: Le terrain est tourné de 6°

- Reprise des algorithmes de Saturne réel : les algorithmes fonctionnent maintenant en simulation, il y avait un problème au niveau du changement d'objectif (pour passer du suivi de ligne au suivi de cercle en autre) car Robin vérifiait que les données du GPS était précise, or en simulation le paramètre n'existe pas et le robot ne changeait pas d'objectif.