Drones

From theory to navigation

https://webperso.ensta.fr/jaulin/drones.html

May 11, 2026

Introduction

A drone is an uncrewed system designed to perform tasks in the physical world without a human operator being physically onboard. It can be remotely controlled, partially autonomous, or fully autonomous, depending on its level of technological sophistication. Unlike the common perception that limits drones to flying devices, the term encompasses a broad class of robotic systems that can operate in the air, on land, on the surface of water, underwater, or even in outer space. What fundamentally characterizes a drone is its ability to sense its environment, make decisions based on those perceptions, and act upon the environment through mechanical or electronic means, either under human supervision.

Drones are used across a wide range of civil, industrial, scientific, and governmental domains. In civil and commercial contexts, they are employed for infrastructure inspection, land surveying, precision agriculture, environmental monitoring, media production, and logistics. Their ability to access hard-to-reach or hazardous areas makes them particularly valuable for inspecting bridges, power lines, pipelines, and offshore structures.

In industrial settings, ground and aerial drones are integrated into warehouses, factories, mining operations, and oil and gas facilities to automate transport, inspection, and monitoring tasks. Scientific research relies heavily on drones for data collection in environments that are dangerous, remote, or inaccessible to humans, such as deep oceans, polar regions, volcanic areas, and extraterrestrial surfaces. In public service and security roles, drones support search and rescue missions, firefighting, disaster assessment, medical supply delivery, law enforcement, and border surveillance. Military and defense applications, while subject to ethical and legal debate, include reconnaissance, logistics, and explosive ordnance disposal.

The technological evolution of drones can be traced from simple remotely controlled machines to sophisticated autonomous systems. Early drones, developed primarily in the twentieth century, relied entirely on direct human control through radio signals and had very limited sensing or decision-making capabilities. These systems were often experimental or military in nature and required continuous operator input.

With the advent of satellite navigation, digital communication, and embedded computing in the late twentieth and early twenty-first centuries, drones gained the ability to follow pre-programmed routes and execute basic automated tasks. Modern drones represent a further evolutionary step, incorporating advanced sensors, real-time data processing, and artificial intelligence. These

technologies enable drones to perceive complex environments, avoid obstacles, adapt to changing conditions, and cooperate with other drones as part of coordinated fleets or swarms. Current research is pushing toward higher levels of autonomy, longer endurance, self-learning capabilities, and seamless integration into human-centered systems such as smart cities and digital infrastructure.

A drone operates through the integration of several core subsystems that function together as a unified whole. Sensors provide the drone with information about its internal state and its external environment. These may include cameras, inertial measurement units, satellite navigation receivers, radar, lidar, sonar, or environmental sensors, depending on the operating domain.

An onboard computer processes this sensor data and executes control algorithms and decision-making logic. This computing unit acts as the drone's brain, determining how it should move, what actions it should take, and how it should respond to both mission objectives and unexpected events. The control system translates these decisions into precise commands that regulate motion and stability.

Actuators, such as motors, propellers, wheels, tracks, or thrusters, physically move the drone and allow it to interact with its surroundings. Communication systems enable the exchange of data and commands between the drone and a remote operator or other drones, while power systems supply the necessary energy, typically through batteries, fuel cells, or hybrid solutions. The continuous feedback loop between sensing, computation, and actuation allows the drone to operate effectively and, in advanced cases, autonomously.

In this lesson; a *drone* can be defined as a mechanical system capable of moving in its environment in an autonomous manner. For that purpose, it must be equipped with:

- *sensors* that will collect knowledge of its surroundings (which it is more or less aware of) and determine its location ;
- *actuators* which will allow it to move ;
- an *intelligence* (or algorithm or *controller*), which will allow it to compute, based on the data gathered by the sensors, the commands to send to the actuators in order to perform a given task.

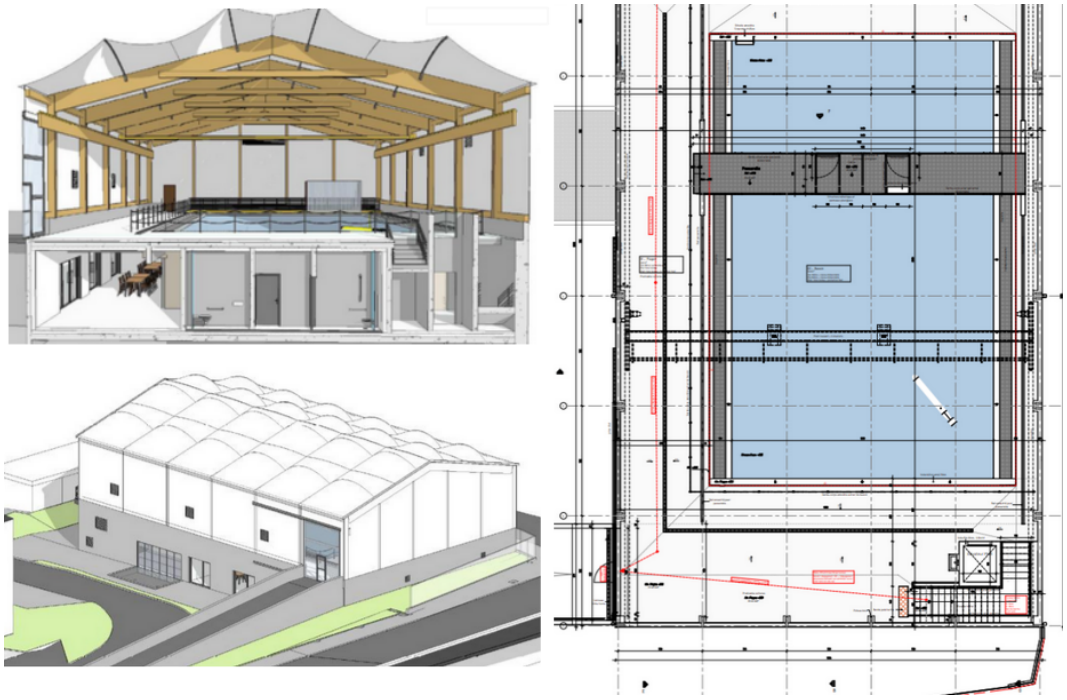
We must add the environment of the robot which corresponds to the world in which it evolves and its *mission* which is the task it has to accomplish.

The aim of this lesson is to give an overview of the tools and methods for the control of drones. The drones that will be used in out experiments are DD-Boats. Such a boat has two propellers. Instead of having a moving rudder, these type of boat turn by slowing down one propeller and speeding up the other. This is called "differential steering," and "DD" is used on the packaging to highlight this Dual Drive system.

In our DD-Boat, the sensors are magnetometers, accelerometers, gyrometers, and a GPS. The actuators are the two propellers (left and right).

DD-Boat (Dual Drive)

These robots are used at the Guerledan lake by the students in Robotics. Here the experiments will take place in the new robotics pool of ENSTA named *Bassin Robotique*.



Pool (Bassin Robotique)

The lesson presents how to control the DD-Boat from the theory to the experiment in the pool. Chapter 1 introduces the fundamentals of nonlinear control. It focuses on the control of a nonlinear mobile robot, with particular emphasis on trajectory and path-following problems. The theoretical tools presented in this chapter form the basis for all subsequent developments. Chapter 2 is dedicated

to the design and implementation of a simulation environment, which will be used to evaluate and validate the proposed control laws under realistic conditions. Chapter 3 explains the calibration of onboard sensors, with special attention given to magnetometers. The magnetometer plays a central role in the navigation, especially for accurate heading estimation when operating inside the pool. Chapter 4 introduces some specific notions of computer vision applied to mobile robotics. Vision-based methods will be used to estimate the actual trajectory of the boat, enabling post-mission analysis and validation of the control objectives. Finally, Chapter 5 provides low-level implementation details required to deploy the controller on the real DD-Boat. Practical aspects such as embedded implementation and hardware constraints are discussed to bridge the gap between theory, simulation, and real-world experimentation.

Chapter 1

Nonlinear control

Drones are strongly nonlinear dynamical systems. In this chapter, we will look at designing nonlinear controllers in order to constrain the state vector of the robot to follow a fixed forward path or to remain within a determined area of its workspace. In contrast to the linear approach, which offers a general methodology but is limited to the neighborhood of a point of the state space [1] [2], nonlinear approaches only apply to limited classes of systems, but they allow to extend the effective operating range of the system. Indeed, there is no general method of globally stabilizing nonlinear systems [3]. However, there is a multitude of methods that apply to particular cases [4] [5]. The aim of this chapter is to present one of the more representative theoretical methods (whereas in the following chapter, we will be looking at more pragmatic approaches). This method is called *feedback linearization* and it requires knowledge of an accurate and reliable state machine for our robot. The robots considered here are mechanical systems whose modeling can be found in [6]. We will assume in this chapter that the state vector is entirely known. In practice, it has to be approximated from sensor measurements.

1.1 Controlling an integrator chain

As we will show further on, feedback linearization leads to the problem of controlling a system which is composed of several integrator chains decoupled from one another. In this paragraph we will therefore consider an integrator chain whose input u and output y are linked together by the differential equation:

$$y^{(n)} = u.$$

This corresponds to a linear system, the control of which is easy, as you have seen in the course of linear control [7].

Let us first of all stabilize this system using a *proportional-derivative* controller of the type:

$$u = \alpha_0 (w - y) + \alpha_1 (\dot{w} - \dot{y}) + \cdots + \alpha_{n-1} \left(w^{(n-1)} - y^{(n-1)} \right) + w^{(n)},$$

where w is the wanted setpoint for y. Let us note that w may depend on time. The fact that this controller requires the differentials of y is not a problem within the frame defined by the feedback linearization. Indeed, all of these derivatives can be described as analytic functions of the state $\mathbf{x}$ of the system and the input $\mathbf{u}$. Concerning the setpoint $w(t)$, it is chosen by the user and an analytic expression of $w(t)$ may be assumed to be known (for instance $w(t) = \sin(t)$). Thus, calculating the differentials of w is done in a formal manner and no sensitivity of the differential operator with respect to the noise will occur.

The feedback system is described by the differential equation:

$$y^{(n)} = u = \alpha_0 (w - y) + \alpha_1 (\dot{w} - \dot{y}) + \cdots + \alpha_{n-1} \left(w^{(n-1)} - y^{(n-1)} \right) + w^{(n)}.$$

If we define the error e between the setpoint w and the output y as $e = w - y$, this equation becomes:

$$e^{(n)} + \alpha_{n-1} e^{(n-1)} + \cdots + \alpha_1 \dot{e} + \alpha_0 e = 0.$$

This differential equation is called the *error dynamics equation*. Its characteristic polynomial, given by:

$$P(s) = s^n + \alpha_{n-1} s^{n-1} + \cdots + \alpha_1 s + \alpha_0, \tag{1.1}$$

can thus be chosen arbitrarily among the polynomials of degree n. Of course, we will choose all roots with a negative real part, in order to ensure the stability of the system. For instance, if $n = 3$ and if we want all the poles to be equal to -1, we will take:

$$s^3 + \alpha_2 s^2 + \alpha_1 s + \alpha_0 = (s+1)^3 = s^3 + 3s^2 + 3s + 1.$$

Whence:

$$\alpha_2 = 3, \alpha_1 = 3, \alpha_0 = 1.$$

The controller obtained is then given by:

$$u = (w - y) + 3 (\dot{w} - \dot{y}) + 3 (\ddot{w} - \ddot{y}) + \dddot{w} - \dddot{y}.$$

Remark 1. For the exercises, we will choose, for simplicity, to position all our poles at -1. The previous reasoning, applied for various degrees n, leads us to the following controls:

$$\begin{array}{ll} n = 1 & u = (w - y) + \dot{w} \\ n = 2 & u = (w - y) + 2 (\dot{w} - \dot{y}) + \ddot{w} \\ n = 3 & u = (w - y) + 3 (\dot{w} - \dot{y}) + 3 (\ddot{w} - \ddot{y}) + \dddot{w} \\ n = 4 & u = (w - y) + 4 (\dot{w} - \dot{y}) + 6 (\ddot{w} - \ddot{y}) + 4 (\dddot{w} - \dddot{y}) + \ddot{\dddot{w}} - \ddot{\dddot{y}}. \end{array} \tag{1.2}$$

Notice that the coefficients correspond to those of Pascal's triangle.

1.2 Introductory example

Before giving the principles of feedback linearization, we will consider an introductory example. Let us take the pendulum of Figure 1.1. The input of this system is the torque u exerted on the pendulum.

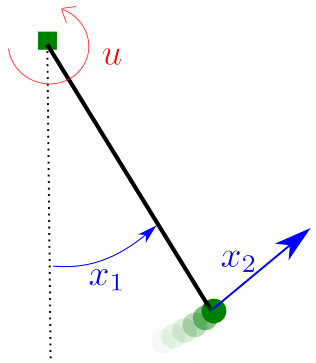


Figure 1.1: Simple pendulum with state vector $\mathbf{x} = (x_1, x_2)$

Its state representation is assumed to be:

$$\left\{ \begin{array}{ccc} \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} & = & \begin{pmatrix} x_2 \\ -\sin x_1 + u \end{pmatrix} \\ y & = & x_1 \end{array} \right.$$

It represents a normalized model in which the coefficients (mass, gravity, length) have all been set to 1. We would like the position $x_1(t)$ of the pendulum to be equal to some setpoint $w(t)$ which may vary over time. By using a feedback linearization method, explained later, we would like to have a state feedback controller such that the error $e = w - x_1$ converges towards 0 as $\exp(-t)$ (which means that we place the poles at -1). Let us differentiate y until the input u appears. We have:

$$\begin{array}{rcl} \dot{y} & = & x_2 \\ \ddot{y} & = & -\sin x_1 + u. \end{array}$$

Let us choose:

$$u = \sin x_1 + v, \tag{1.3}$$

where v corresponds to the new, so-called intermediate input. We obtain

$$\ddot{y} = v. \tag{1.4}$$

Such a feedback is called *linearizing feedback* because it transforms the nonlinear system into a linear system. The system obtained in this way can be stabilized by standard linear techniques. Let us

take as an example a proportional-derivative controller:

$$\begin{array}{rcl} v & = & (w-y)+2\,(\dot{w}-\dot{y})+\ddot{w} \\ & = & (w-x_1)+2\,(\dot{w}-x_2)+\ddot{w}. \end{array}$$

By injecting this expression of v into (1.4), we obtain:

$$\ddot{y} = (w-x_1)+2\,(\dot{w}-x_2)+\ddot{w}.$$

Which yields:

$$e+2\dot{e}+\ddot{e}=0$$

where $e = w - x_1$ is the error between the position of the pendulum and its setpoint. The complete controller is expressed by:

$$u \overset{(1.3)}{=} \sin x_1 + (w-x_1)+2\,(\dot{w}-x_2)+\ddot{w}.$$

If we now want the angle x_1 of the pendulum to be equal to $\sin t$ once the transient regime has passed, we simply need to take $w(t) = \sin t$. Thus, $\dot{w}(t) = \cos t$ and $\ddot{w} = -\sin t$. Consequently, the controller is given by:

$$u = \sin x_1 + (\sin t - x_1) + 2\,(\cos t - x_2) - \sin t.$$

In this very simple example, we can see that the proposed controller is nonlinear and depends on time. No approximation arising from linearization has been performed. Of course, a linearization has been done by a first feedback in order to make the system linear, but this linearization did not introduce any approximation.

1.3 Dubins car

Consider a robot (a *Dubins car* [8]) described by the state equations:

$$\left\{ \begin{array}{rcl} \dot{x}_1 & = & x_4 \cdot \cos x_3 \\ \dot{x}_2 & = & x_4 \cdot \sin x_3 \\ \dot{x}_3 & = & u_1 \\ \dot{x}_4 & = & u_2 \end{array} \right.$$

where x_4 is the speed of the car, x_3 its orientation and (x_1, x_2) the coordinates of its center (see Figure 1.2).

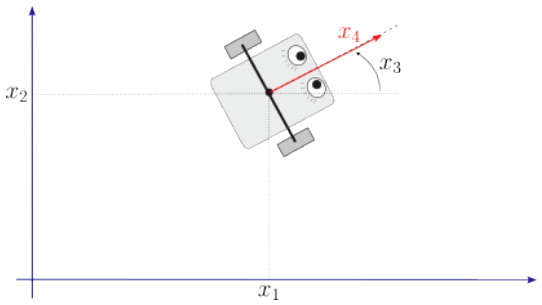


Figure 1.2: Dubins car

The state vector is $\mathbf{x} = (x_1, x_2, x_3, x_4)$. We would like to compute a controller for the robot to describe the Lissajou trajectory:

$$\left\{ \begin{array}{rcl} w_1(t) & = & \sin t \\ w_2(t) & = & \cos 2t \end{array} \right.$$

For this, we use a feedback linearizing method. We define the output

$$\mathbf{y} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

We have:

$$\begin{array}{rcl} \dot{\mathbf{y}} & = & \begin{pmatrix} x_4 \cdot \cos x_3 \\ x_4 \cdot \sin x_3 \end{pmatrix} \\ \ddot{\mathbf{y}} & = & \begin{pmatrix} u_2 \cos x_3 - u_1 x_4 \sin x_3 \\ u_2 \sin x_3 + u_1 x_4 \cos x_3 \end{pmatrix} = \underbrace{\begin{pmatrix} -x_4 \sin x_3 & \cos x_3 \\ x_4 \cos x_3 & \sin x_3 \end{pmatrix}}_{\mathbf{A}(\mathbf{x})} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \end{array}$$

If we take as input:

$$\mathbf{u} = \mathbf{A}^{-1}(\mathbf{x}) \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

where $\mathbf{v} = (v_1, v_2)$ is the new input vector, we obtain the linear system:

$$\ddot{\mathbf{y}} = \mathbf{v}.$$

Let us transform this system so that all our poles are at -1. Following (1.2), we obtain:

$$\begin{array}{rcl} \mathbf{v} & = & (\mathbf{w} - \mathbf{y}) + 2 \cdot (\dot{\mathbf{w}} - \dot{\mathbf{y}}) + \ddot{\mathbf{w}} \\ & = & \left(\mathbf{w} - \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right) + 2 \cdot \left(\dot{\mathbf{w}} - \begin{pmatrix} x_4 \cos x_3 \\ x_4 \sin x_3 \end{pmatrix} \right) + \ddot{\mathbf{w}} \end{array}$$

If we define the error vector $\mathbf{e} = (e_1, e_2) = \mathbf{w} - \mathbf{y}$, the error dynamics are written as:

$$\mathbf{e} + 2\dot{\mathbf{e}} + \ddot{\mathbf{e}} = 0$$

which is stable and quickly converges towards 0. The controller will therefore be:

$$\mathbf{u} = \mathbf{A}^{-1}(\mathbf{x}) \cdot \left(\left(\mathbf{w} - \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right) + 2 \cdot \left(\dot{\mathbf{w}} - \begin{pmatrix} x_4 \cos x_3 \\ x_4 \sin x_3 \end{pmatrix} \right) + \ddot{\mathbf{w}} \right) \tag{1.5}$$

with

$$\begin{aligned} \mathbf{w} &= \begin{pmatrix} \sin t \\ \cos 2t \end{pmatrix} \\ \dot{\mathbf{w}} &= \begin{pmatrix} \cos t \\ -2 \sin 2t \end{pmatrix} \\ \ddot{\mathbf{w}} &= \begin{pmatrix} -\sin t \\ -4 \cos 2t \end{pmatrix} \end{aligned}$$

1.4 Kinematic model and dynamic model

1.4.1 Principle

The dynamic model for the robot are of the form:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$$

where $\mathbf{u}$ is the vector of the external forces (that are under our control). The function $\mathbf{f}$ involves dynamic coefficients (such as masses, inertial moments, coefficients of friction, etc.) as well as geometric coefficients (such as lengths). The dynamic coefficients are generally not well known and can change in time with wear or usage. If we now take as new input the vector $\mathbf{a}$ of the desired accelerations at the application points of the forces (in the direction of the forces), we obtain a new model, referred to as *kinematic*, of the form:

$$\dot{\mathbf{x}} = \boldsymbol{\varphi}(\mathbf{x}, \mathbf{a})$$

but in this new model most of the dynamic coefficients have disappeared. It is possible to switch from a dynamic model to a kinematic model using a so-called *high-gain* controller, of the form:

$$\mathbf{u} = K \cdot (\mathbf{a} - \mathbf{a}_m)$$

where K is a very large real number and $\mathbf{a}_m$ is the vector of measured accelerations. It depends on both $\mathbf{x}$, $\mathbf{u}$, *i.e.*, there exists a function $\mathbf{h}$ such that

$$\mathbf{a}_m = \mathbf{h}(\mathbf{x}, \mathbf{u})$$

Thus, we have:

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}, K \cdot (\mathbf{a} - \mathbf{h}(\mathbf{x}, \mathbf{u}))\right) \Leftrightarrow \dot{\mathbf{x}} = \boldsymbol{\varphi}(\mathbf{x}, \mathbf{a}).$$

as illustrated by Figure 1.3. If K is large enough, we can assume that $\mathbf{a} = \mathbf{a}_m$.

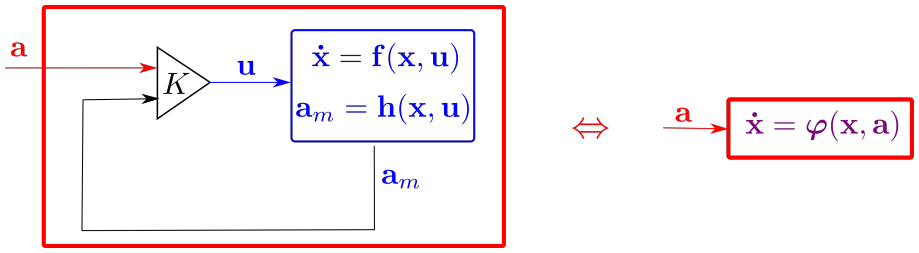


Figure 1.3: The loop transforms a dynamic model into a kinematic model

The simple high-gain feedback has allowed us to get rid of numerous dynamic parameters and switch from an uncertain system to a reliable one, with well-known geometric coefficients. This high-gain feedback is known in electronics as an operational amplifier, where it is used with the same idea of robustness.

Switching from a dynamic model to a kinematic one has the following advantages:

- the linearizing controller developed in this chapter requires using a reliable model such as a kinematic model. If the coefficients (which are not measured) are not well known (such as in the case of dynamic systems), the linearizing controller will not work in practice ;
- the kinematic model is easier to put into equations. It is not necessary to have a dynamic model to obtain the latter ;
- the servo-motors (see paragraph 1.4.2) are cheap and easy to find. They incorporate this high-gain controller. We may see them as mechanical operational amplifiers.

We will illustrate the concept in the following paragraph, through the example of the inverted rod pendulum.

1.4.2 Servo-motors

A mechanical system is controlled by forces or torques and obeys a dynamic system that depends on numerous little-known coefficients. This same mechanical system represented by a kinematic model is controlled by positions, speeds or accelerations. The kinematic model depends on well-known geometric coefficients and is much simpler to put into equations. In practice, one switches from a dynamic model to its kinematic equivalent by adding servo-motors. In summary, a servo-motor is a DC motor with an electrical control circuit and a sensor (of position, speed or acceleration). The control circuit calculates the voltage u to give the motor in order for the quantity measured by the sensor to correspond to the setpoint w. There are three types of servo-motors:

- the *position servo*. The sensor measures the position (or the angle) x of the motor and the control law is expressed by $u = K(x - w)$. If K is large, we may conclude that $x \simeq w$;
- the *speed servo*. The sensor measures the speed (or the angular speed) $\dot{x}$ of the motor and the control law is expressed by $u = K(\dot{x} - w)$. If K is large, we have $\dot{x} \simeq w$;
- the *acceleration servo*. The sensor measures the acceleration (tangential or angular) $\ddot{x}$ of the motor and the control law is expressed by $u = K(\ddot{x} - w)$. If K is large, we have $\ddot{x} \simeq w$. It is this type of servo-motor that we have chosen for the inverted rod pendulum.

Thus, when we wish to control a mechanical system, the use of servo-motors allows us (i) to have a model that is easier to obtain, (ii) to have a model with fewer coefficients that is closer to reality and (iii) to have a more robust controller with respect to any modification of the dynamic coefficients of the system.

1.5 Model-free control

When we implement a controller for a robot and perform the initial tests we rarely succeed on the first try, which leads us to the problem of debugging. It might be that the compass is subject to electromagnetic disturbances, that it is placed upside-down, that there is a unit conversion problem in the sensors, that the motors are saturated or that there is a sign problem in the equations of the controller. The problem of debugging is a complex one and it is wise to respect the *continuity principle*: each step in the construction of the robot must be of reasonable size and has to be validated before pursuing construction. Thus, for a robot, it is desirable to implement a simple intuitive controller that is easy to debug before setting up a more advanced one. This principle can not always be applied. However, if we have a good *a priori* understanding of the control law to apply, then such a continuity principle can be followed. Among mobile robots for which a pragmatic controller can be imagined, we can distinguish at least two sub-classes:

- *vehicle-robots*. These are systems built by man to be controlled by man such as the bicycle, the sailboat, the car, etc. We will try to copy the control law used by humans and transform it into an algorithm ;
- *biomimetic robots*. These robots are inspired by the movement of human beings. We have been able to observe them for long periods of time and deduce the strategy developed by nature to design its control law. This is the *biomimetic* approach (see for example [9]). We do not include walking robots in this category because, even though we all know how to walk, it is near to impossible to know which control law we use for it. Thus, designing a control law for walking robots [10] can not be done without a complete mechanical modeling of walking and without using any theoretical automatic control methods such as those evoked in the previous chapter.

For these two classes, we often do not have simple and reliable models available (this is the case for example of the sailboat or the bicycle). However, the strong understanding we have of them will allow us to build a robust control law.

We show, using several examples, how to design such control laws. These will be referred to as *mimetic control* (we are trying to imitate humans or animals) or *model-free control* (we do not use the state equations of the robot to design the controller). Although model-free approaches have been largely explored in theory, here we will use the intuition we have of the functioning of our robot as much as possible.

Model-free control of a Dubins car

In order to illustrate the principle of model-free control, let us consider the case of a Dubins car described by:

$$\begin{cases} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= u_2 \\ \dot{v} &= u_1 - v \end{cases}$$

This model can be used for simulation, but not for obtaining the controller.

Proportional heading and speed controller

We will now propose a simple controller for this system by using our intuition about the system. Let us take $\widetilde{\theta} = \theta_d - \theta$ where θ_d is the desired heading and $\widetilde{v} = v_d - v$ where v_d is the desired speed.

- For speed control, we take:

  $$u_1 = a_1 \tanh \widetilde{v}$$

  where a_1 is a constant representing the maximum acceleration (in absolute value) that the motor is able to deliver. The hyperbolic tangent *tanh* (see Figure 1.4) is used as saturation function. Let us recall that :

  $$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.6}$$

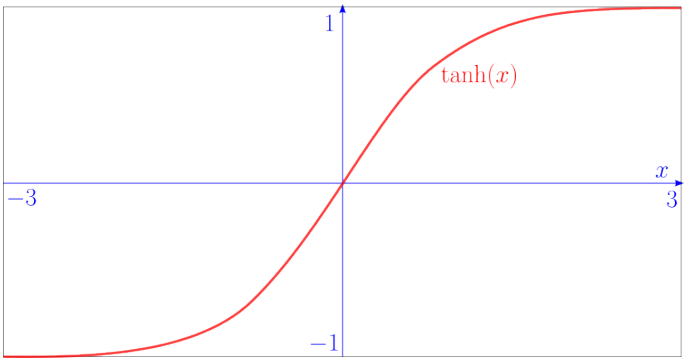


Figure 1.4: Hyperbolic tangent function used as saturation function

- For the heading control, we take:

$$u_2 = a_2 \cdot \text{sawtooth}(\widetilde{\theta})$$

In this last formula, *sawtooth* $\mathbb{R} \to \mathbb{R}$ corresponds to the sawtooth function which is the identity in $[-\pi, \pi]$ and 2π-periodic. It is defined by:

$$\text{sawtooth}(\widetilde{\theta}) = 2\text{atan}\left(\tan\frac{\widetilde{\theta}}{2}\right) = \text{mod}(\widetilde{\theta} + \pi, 2\pi) - \pi \tag{1.7}$$

Let us note that for numerical reasons, it is preferable to use the expression containing the *modulus* function. As illustrated in Figure 1.5, the function corresponds to an error in heading. The interest in taking an error $\widetilde{\theta}$ filtered by the *sawtooth* function is to avoid the problem of the $2k\pi$ modulus: we would like a $2k\pi$ to be considered non-zero.

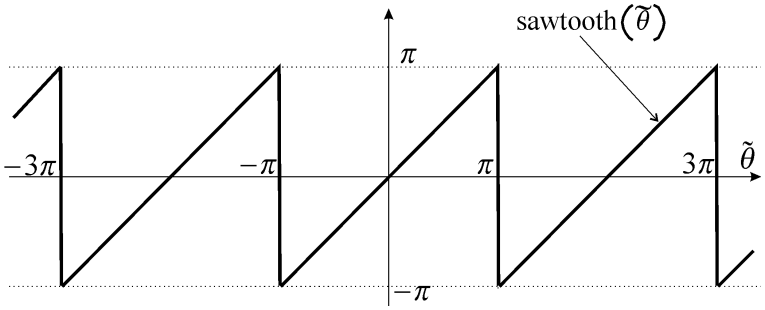


Figure 1.5: *Sawtooth* function used to avoid jumps in the heading control

The controller is thus defined by:

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} a_1 \cdot \tanh(v_d - v) \\ a_2 \cdot \text{sawtooth}(\theta_d - \theta) \end{pmatrix}$$

This model-free control does not need to use the state equations of the robot. It is based on the understanding that we have of the dynamics of the system and recalls our wireless operation method

of the Dubins car. It has two parameters a_1 and a_2 that are easy to set (a_1 represents the propelling power and a_2 the directional disturbances). Finally, this controller is easy to implement and to debug.

1.6 Navigation

Once a line tracking has been correctly implemented and validated, a number of lines should be chained together with the aim of performing complex missions (such as for instance connecting two points of the globe). In such a context, a Petri net strategy is well-adapted for representing the discrete state changes [11]. Before the robot is launched, it is in an initial state represented by the place p_0, as illustrated by Figure 1.6. The transition t_1 is crossed at the start of the mission. If everything goes well, the robot is in state p_1 and is tracking its first line $\mathbf{a}_1\mathbf{b}_1$.

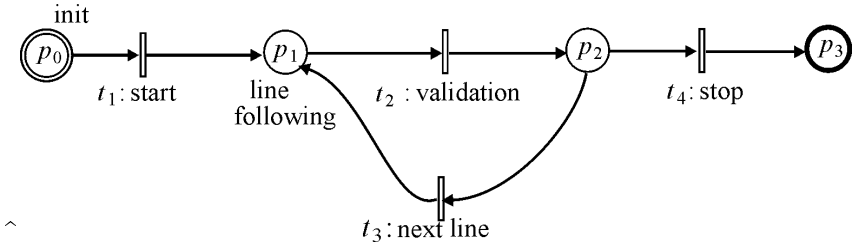


Figure 1.6: Petri net supervising the navigation of the drone

The line **ab** is validated as soon as the point **b** is surpassed, *i.e.*, if $\langle \mathbf{b}-\mathbf{a}, \mathbf{m}-\mathbf{b} \rangle > 0$ or equivalently,

$$(b_1 - a_1)(m_1 - b_1) + (b_2 - a_2)(m_2 - b_2) > 0$$

as illustrated by Figure 1.7.

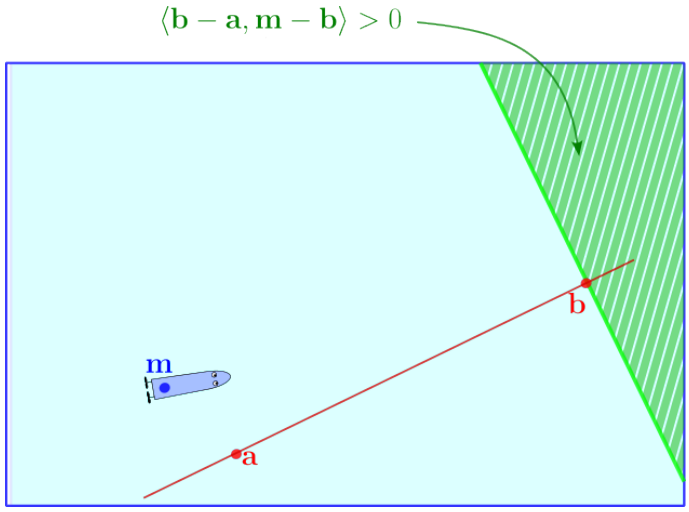


Figure 1.7: The DD-Boat validates the line **ab** as soon as it enters in the green half plane

Once a line is validated, we proceed to the next line. When the list of lines to track is empty, the mission ends (place p_3).

Exercises

Exercise 1.– *Crank*

Let us consider the manipulator robot, or *crank* of Figure 1.8 (on the left). This robot is composed of two arms of length ℓ_1 and ℓ_2. Its two degrees of freedom, denoted by x_1 and x_2, are represented on the figure. The inputs u_1, u_2 of the system are the angular speeds of the arms (*i.e.*, $u_1 = \dot{x}_1, u_2 = \dot{x}_2$). We will take as output the vector $\mathbf{y} = (y_1, y_2)$ corresponding to the end of the second arm.

1) Give the state equations of the robot. We will take the state vector $\mathbf{x} = (x_1, x_2)$.

2) We would like $\mathbf{y}$ to follow a setpoint $\mathbf{w}$ describing a target circle (on the right of Figure 1.8). This setpoint satisfies:

$$\mathbf{w} = \mathbf{c} + r \cdot \begin{pmatrix} \cos t \\ \sin t \end{pmatrix}.$$

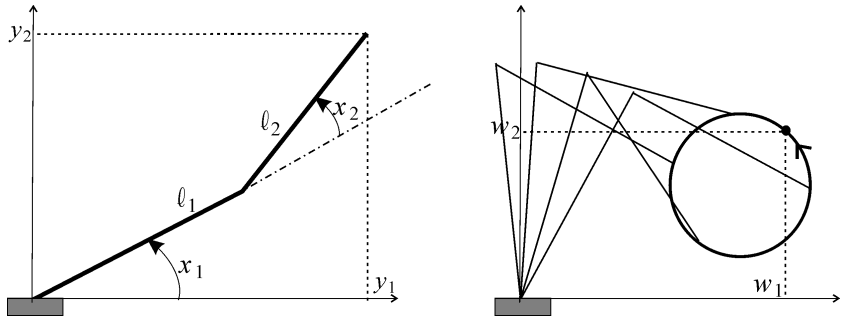


Figure 1.8: Manipulator robot whose end effector must follow a circle

Give the expression of a control law that allows to perform this task. We will use a feedback linearization method and we will place the poles at -1.

3) Study the singularities of the control.

4) Let us consider the case $\ell_1 = \ell_2$, $\mathbf{c} = (3, 4)$ and $r = 1$. For which values of ℓ_1 are we certain to be able to move freely on the target circle, without encountering singularities ?

5) Write a Python program illustrating this control law. The starter code named `crank.py` has to be downloaded.

EXERCISE 2.– *Train*

Let us consider a robot A (the DD-Boat of Figure 1.9) described by the following state equations:

$$\begin{cases} \dot{x}_a &= v_a \cos\theta_a \\ \dot{y}_a &= v_a \sin\theta_a \\ \dot{\theta}_a &= u_{a1} \\ \dot{v}_a &= u_{a2} \end{cases}$$

where v_a is the speed of the robot, θ_a its orientation and (x_a, y_a) the coordinates of its center. The robot is able to measure its state variables with very high precision.

1) Calculate $\ddot{x}_a, \ddot{y}_a$ with respect to $x_a, y_a, v_a, \theta_a, u_{a1}, u_{a2}$.

2) Propose a controller that allows to follow the trajectory:

$$\begin{cases} \hat{x}_a(t) &= 3\sin(\omega t) + 6 \\ \hat{y}_a(t) &= 5\cos(\omega t) + 10 \end{cases}$$

with $\omega = 0.1$. A feedback linearization method must be used for this.

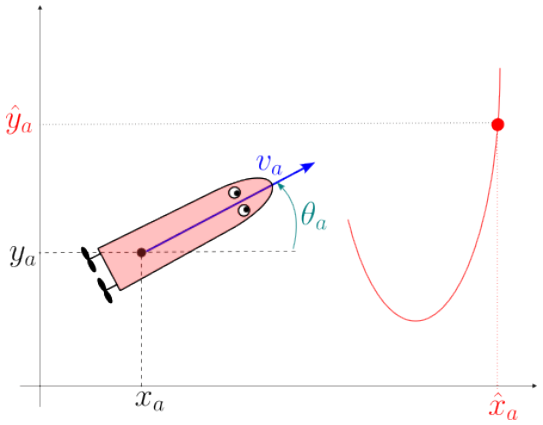


Figure 1.9: Our DD-Boat (red) following a target point $(\hat{x}_a, \hat{y}_a)$

3) A second DD-Boat B of the same type as A wishes to follow DD-Boat A (see Figure 1.10). We define a target point with coordinates $(\hat{x}_b, \hat{y}_b)$ for DD-Boat B (green) behind A. DD-Boat A sends to B information associated with the target point wirelessly.

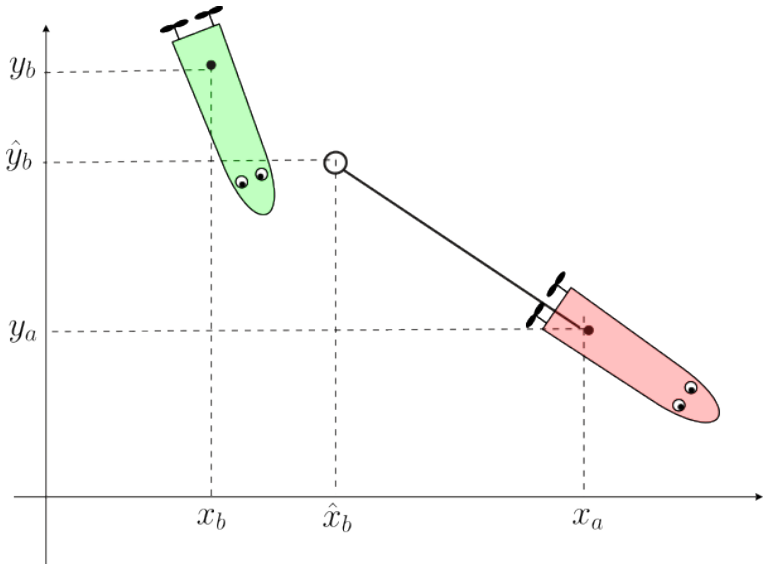


Figure 1.10: DD-Boat B (green) has to follow DD-Boat A (red)

This point will be positioned at the rear of DD-Boat A at a distance ℓ of our reference point (x_a, y_a). Give the expression of these quantities with respect to the state variables of DD-Boat A.

4) Simulate in Python this second vehicle B together with its controller following A. The starter code named `train.py` has to be downloaded.

5) Add a third DD-Boat C that follows B with the same principle. Simulate the entire system. Figure 1.11 illustrates the behavior you should get for the robots. DD-Boat A follows the ellipse, B follows A and C follows B.

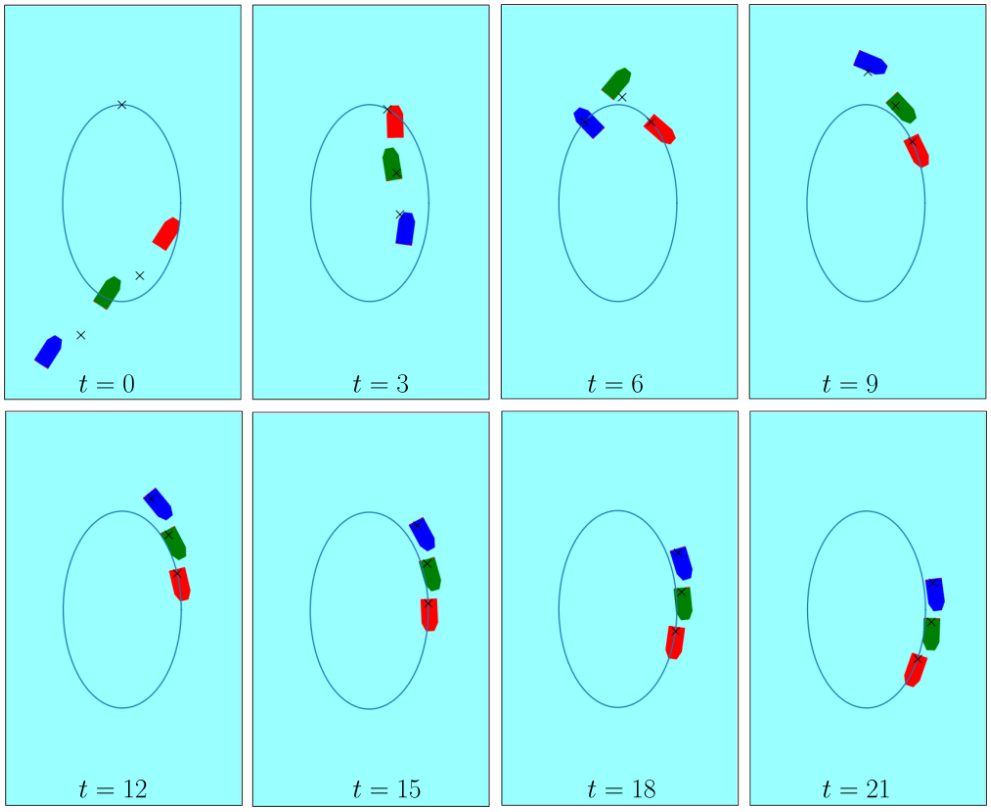


Figure 1.11: Illustration of the execution of the program ; Robot A (red) is followed by B (green) which is followed by C (blue) ; the setpoints and attachment points are represented by $\times$

EXERCISE 3.– *Line following*

Let us consider a DD-Boat moving at the surface of a pool, the shape of which is the rectangle $[0,12] \times [0,20]$. The boat is described by the following state equations:

$$\begin{cases} \dot{x}_1 &= \cos\theta \\ \dot{x}_2 &= \sin\theta \\ \dot{x}_3 &= u \end{cases}$$

where x_3 is the heading of the robot and $\mathbf{m} = (x_1, x_2)$ are the coordinates of its center. The state vector is given by $\mathbf{x} = (x_1, x_2, x_3)$.

1) Propose a heading controller for the DD-Boat.

2) Give a controller for the robot to track the line **ab**. This line must be attractive. Give the condition which tells us that the point **b** is overtaken.

3) In Python, make a program which simulates the robot tracking a closed path composed of a sequence of lines $\mathbf{a}_j\mathbf{b}_j$, $j \in \{0, \ldots, 3\}$, where

$$\begin{array}{lllllll} \mathbf{a}_0 & = & (8,4) & \mathbf{b}_0 & = & (8,16) \\ \mathbf{a}_1 & = & (8,16) & \mathbf{b}_1 & = & (4,16) \\ \mathbf{a}_2 & = & (4,16) & \mathbf{b}_2 & = & (4,4) \\ \mathbf{a}_3 & = & (4,4) & \mathbf{b}_3 & = & (8,4) \end{array}$$

The resulting behavior you should obtain is illustrated by Figure 1.12. The initial pose ($\mathbf{x}(0) = (0,0,0)$) corresponds to the bottom left corner. The robot first follows the line $\mathbf{a}_0\mathbf{b}_0$ (left), then it follows the line $\mathbf{a}_1\mathbf{b}_1$, etc. The starter code is named `line_follow.py`.

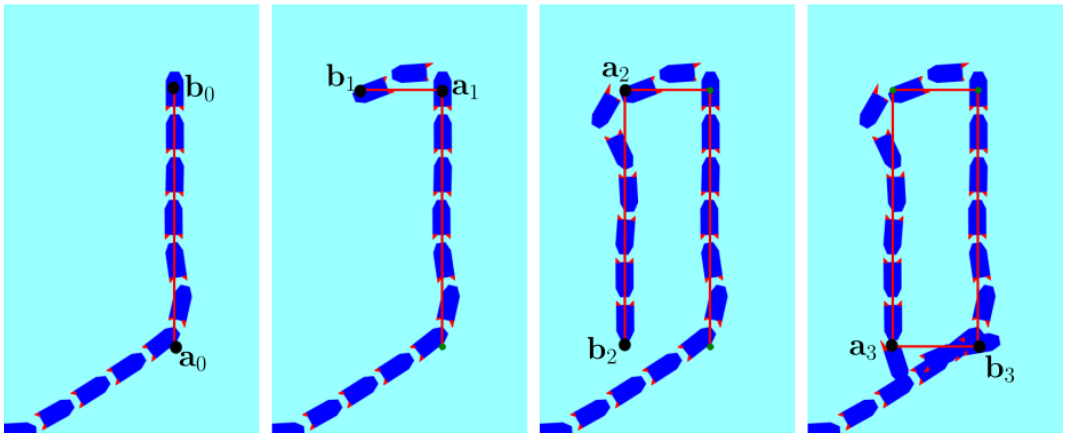


Figure 1.12: Illustration of the execution of the program. The DD-Boat follows a rectangle of the pool side by side

Chapter 2

Simulator

In this chapter, we propose a nonlinear state-space model of the DD-Boat, seen as a surface vessel equipped with two propellers (left and right). The model is intended for control and estimation purposes and is commonly used in marine robotics. The vessel motion is described in the horizontal plane using three degrees of freedom: surge, sway and yaw.

A state space model should have the form:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \\ \mathbf{y}(t) = \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)) \end{cases}$$

under the hypothesis that the time t in which the system evolves is continuous. The vector $\mathbf{u}(t)$ is the *input* (or *control*) of the system. Its value may be chosen arbitrarily for all t. The vector $\mathbf{y}(t)$ is the *outputs* of the system that be measured with a certain degree of accuracy. The vector $\mathbf{x}(t)$ is called the *state* of the system. It represents the memory of the system, in other words the information needed by the system in order to predict its own future, for a known input $\mathbf{u}(t)$. The first of the two equations is called the *evolution equation*. It is a differential equation which enables us to know where the state $\mathbf{x}(t)$ is headed knowing its value at the present moment t and the control $\mathbf{u}(t)$ which we are currently exerting. The second equation is called the *observation equation*. It allows us to compute the output vector $\mathbf{y}(t)$, knowing the state and the control at time t. Note however that, unlike the evolution equation, this equation is not a differential equation since it does not involve the derivatives of the signals. The two equations above form the *state representation* of the system.

The Euler or the Runge-Kutta integration methods are often used for simulating the evolution of the system. They is given by:

$$\begin{array}{rcll} \mathbf{x}(t+dt) & = & \mathbf{x}(t) + dt.\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) & \text{(Euler)} \\ \mathbf{x}(t+dt) & = & \mathbf{x}(t) + dt.\mathbf{f}\left(\mathbf{x}(t) + \frac{dt}{2}\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \mathbf{u}(t+\frac{dt}{2})\right) & \text{(second order Runge Kutta)} \end{array}$$

where dt is a small integration time.

2.1 State and Input Variables

We propose a flat two-dimensional model [12] for the DD-Boat where the state vector is defined as: $\mathbf{x} = (x, y, \theta, v_x, v_y, \omega, \omega_1, \omega_2)^{\mathrm{T}}$ with:

- x, y: position in the inertial frame
- θ: heading
- v_x: surge velocity
- v_y: sway velocity
- ω: yaw rate
- ω_1, ω_2: angular speeds of the left and right propellers

The control input is:

$$\mathbf{u} = (u_1, u_2)^{\mathrm{T}}$$

where u_1 and u_2 are the voltages applied to the left and right propellers, as by Figure 2.1.

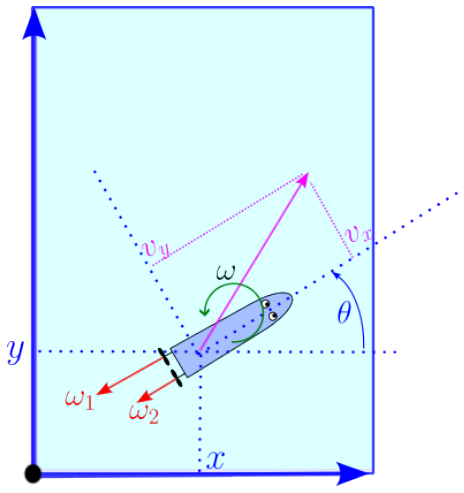


Figure 2.1: A DD-Boat in the pool

2.2 Evolution equations

A nonlinear model for the evolution is:

$$\begin{array}{cccc}
(i) & \dot{x} & = & v_x \cos\theta - v_y \sin\theta \\
(ii) & \dot{y} & = & v_x \sin\theta + v_y \cos\theta \\
(iii) & \dot{\theta} & = & \omega \\
(iv) & \dot{v}_x & = & \omega v_y - p_5 v_x |v_x| + p_3(\omega_1|\omega_1| + \omega_2|\omega_2|) \\
(v) & \dot{v}_y & = & -\omega v_x - p_6 v_y |v_y| \\
(vi) & \dot{\omega} & = & -p_7 \omega|\omega| + p_4(\omega_2|\omega_2| - \omega_1|\omega_1|) \\
(vii) & \dot{\omega}_1 & = & -p_1\omega_1|\omega_1| + p_2 u_1 \\
(viiii) & \dot{\omega}_2 & = & -p_1\omega_2|\omega_2| + p_2 u_2
\end{array}$$

Equations (i), (ii), (iii) correspond to the kinematics of the DD-Boat. Equations (iv), (v) are a direct consequence of the *transport theorem* (or Bour's formula) in mechanics which states that the time derivative of a vector in a rotating frame equals its derivative in that frame plus the cross product with the frame's angular velocity. Equation (vi) correspond to the Newton law for rotating objects. Equations (vii), (viii) assume that the speed of each propeller is modeled using a first-order nonlinear dynamics. The parameters have the following meaning: p_1 represents hydrodynamic and mechanical losses, p_2 is the actuator gain, p_3 is the propulsion gain, p_4 is the rotation gain, p_5 is surge friction (*cavalement* in French), p_6 is sway friction (*embardée* in French) and p_7 is yaw friction.

2.3 GNSS

GNSS (Global Navigation Satellite System) refers to any satellite constellation that provides signals from space, transmitting positioning and timing data to receivers on Earth. GPS is just one specific type of GNSS. It is owned and operated by the U.S. Space Force. Because it was the first system available for global civilian use, the name GPS is now used to describe the entire technology. Standard GPS calculates the position by measuring how long it takes a signal to travel from a satellite to the receiver.

In this course, GPS can be seen as a sensor which returns the position of the DD-Boat. This section describes some notions to be understood to use a GPS such as how to transform a position from a global frame to a local frame. It also explains the principle of multilateration to get a position from the satellites.

2.3.1 Transformations

For long paths over the surface of the Earth, the Cartesian coordinate system, which assumes a flat Earth, can no longer be considered. We then have to rethink our control laws by navigating

relative to a spherical coordinate system (also referred to as *geographical coordinates*), which rotates together with the Earth. Let us denote by ℓ_x the longitude and by ℓ_y the latitude of the point being considered. The transformation in the geographical coordinate system is written as:

$$\mathcal{T}: \begin{pmatrix} \ell_x \\ \ell_y \\ \rho \end{pmatrix} \to \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \rho \cos \ell_y \cos \ell_x \\ \rho \cos \ell_y \sin \ell_x \\ \rho \sin \ell_y \end{pmatrix} \tag{2.1}$$

When $\rho = 6\ 370$ km, we are on the surface of the Earth, which we will assume to be spherical.

Let us consider two points **a**, **m** on the surface of the Earth. Assuming that the two points **a**, **m** are not too far apart (by no more than 100 km), we may consider being in the plane and use a local map, as shown on Figure 2.2.

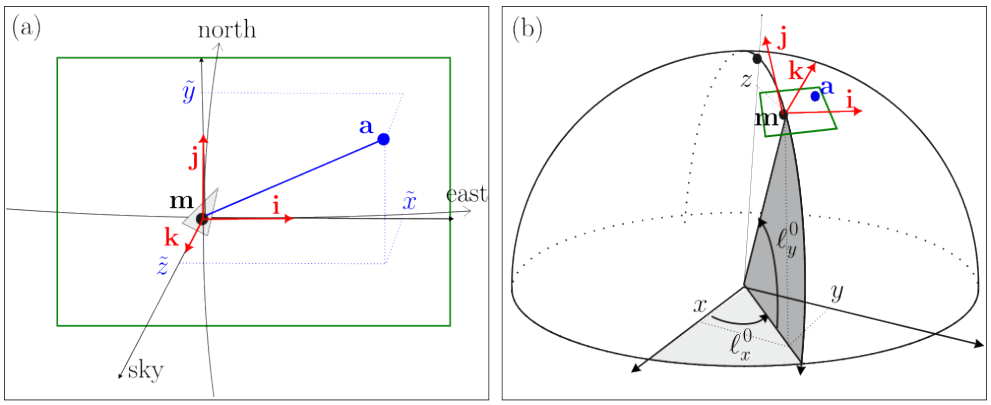


Figure 2.2: (a) The local map with origin **m** ; (b) The local map (green) on the Earth

From latlong to a local map. Let us take a local map $\mathcal{R}_{\mathbf{m}}$ centered on a point **m**, *i.e.*, a coordinate system located on the surface of the Earth whose origin is **m** and whose directions are East-North-Up. We have

$$\begin{pmatrix} \widetilde{x} \\ \widetilde{y} \\ \widetilde{z} \end{pmatrix} \simeq \begin{pmatrix} \rho \cdot \cos \ell_y^0 \cdot (\ell_x - \ell_x^0) \\ \rho \cdot (\ell_y - \ell_y^0) \\ \rho - \rho^0 \end{pmatrix}.$$

This transformation is known as ENU (East, North, Up) projection. An other variant exists (not used here) is NED (North, East, Down). Both are commonly used in aviation and marine cybernetics.

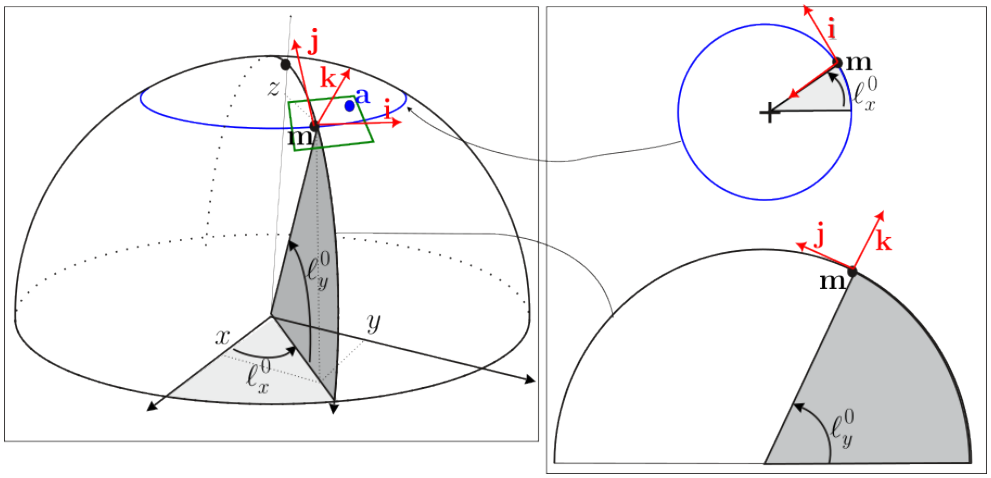


Figure 2.3: Getting Cartesian coordinates in a local frame centred in **m**

For the Robotics pool, the reference point **m** is given using the NMEA uses a specific format called Degrees, Decimal Minutes (DDM): $4°28.162682'W$ and $48°25.154359'N$ (see Figure 2.4). To get into radian we can run the following Python code:

```
# lat0, lon0 = 4825.154359, 428.162682
# Conversion NMEA DDMM.MMMM -> DDM -> Radians
lat0=np.radians(48+25.154359/60) # approx 0.8450751
lon0 = np.radians(-(4+28.162682/60)) # approx -0.0780054
```

We get

$$\begin{aligned} \ell_x^0 &= -0.0780054[\text{rad}] \\ \ell_y^0 &= 0.8450751[\text{rad}] \end{aligned}$$

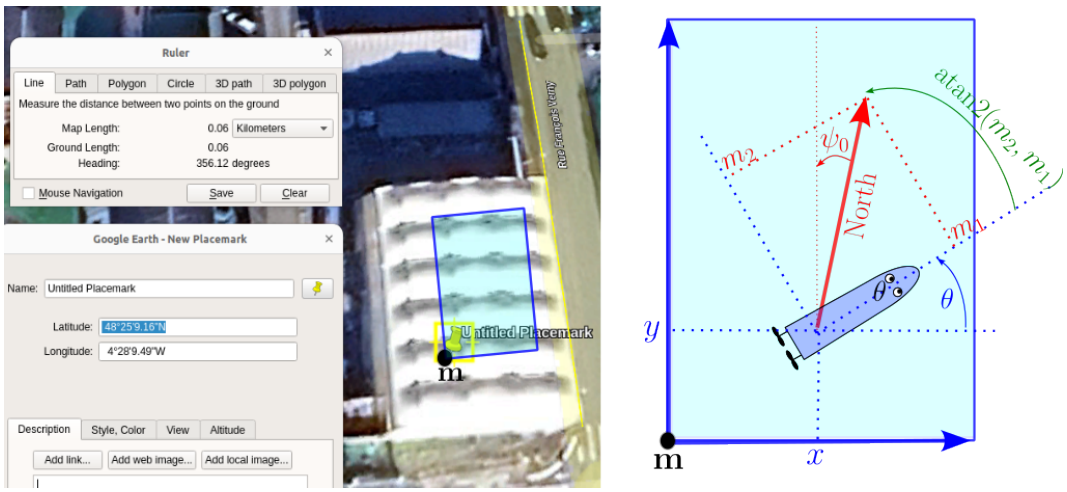


Figure 2.4: Pool represented in Google Earth

To fit the pool, we also have to rotate by an angle $\psi_0 = -0.0677$[rad]. We get

$$\begin{pmatrix} \widetilde{x} \\ \widetilde{y} \end{pmatrix} = \begin{pmatrix} \cos\psi_0 & -\sin\psi_0 \\ \sin\psi_0 & \cos\psi_0 \end{pmatrix} \cdot \begin{pmatrix} \rho \cdot \cos\ell_y^0 \cdot (\ell_x - \ell_x^0) \\ \rho \cdot (\ell_y - \ell_y^0) \end{pmatrix} \tag{2.2}$$

From local map to latlong. We invert the formula we had to convert the latlong coordinates to the local map. We get

$$\begin{pmatrix} \cos\psi_0 & \sin\psi_0 \\ -\sin\psi_0 & \cos\psi_0 \end{pmatrix} \cdot \begin{pmatrix} \widetilde{x} \\ \widetilde{y} \end{pmatrix} \simeq \begin{pmatrix} \rho \cdot \cos\ell_y^0 \cdot (\ell_x - \ell_x^0) \\ \rho \cdot (\ell_y - \ell_y^0) \end{pmatrix}$$

i.e.,

$$\begin{pmatrix} \ell_x \\ \ell_y \end{pmatrix} = \begin{pmatrix} \ell_x^0 + \frac{\cos\psi_0\widetilde{x} + \sin\psi_0\widetilde{y}}{\rho \cdot \cos\ell_y^0} \\ \ell_y^0 + \frac{-\sin\psi_0\widetilde{x} + \cos\psi_0\widetilde{y}}{\rho} \end{pmatrix}.$$

2.3.2 Multilateration

In this lesson, localization by GPS is seen has a magic bloc which returns the position on the Earth (latitude, longitude and altitude). Its principle relies on a localization technique name *multilateration* that is briefly recalled in this section.

The principle of multilateration is to measure the difference of the distances between the robot and the landmarks (the satellites). Indeed, in a number of situations (such as in GNSS localization, as illustrated by Figure 2.5), the clocks between the landmarks and the robot are not synchronized and we can not therefore directly measure the absolute distance between the landmarks and the robot (by the propagation time of air- or soundwaves), but we can measure the difference between these distances. We will now give the principle of this technique.

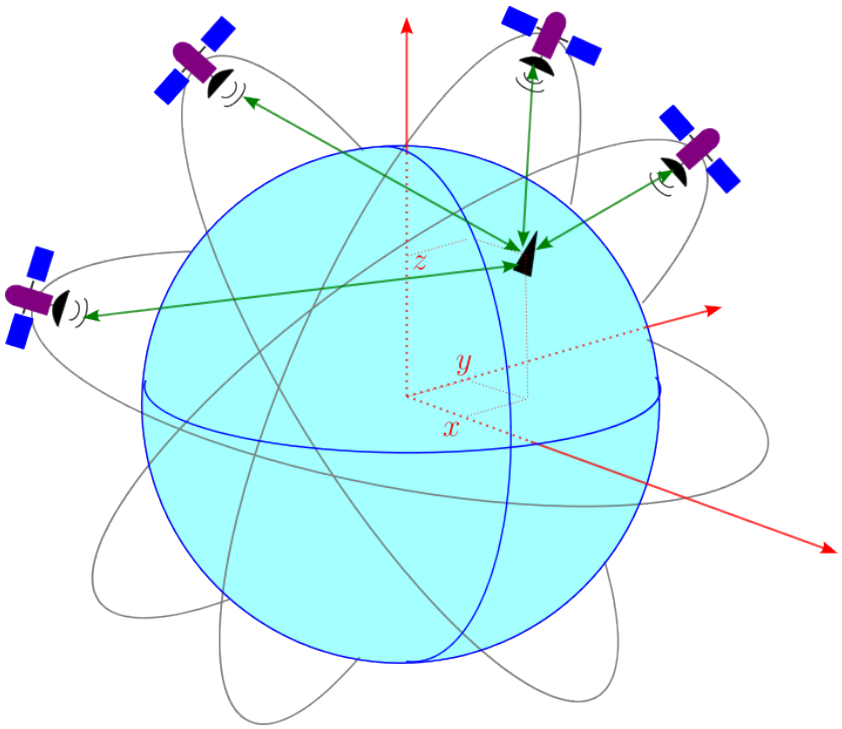


Figure 2.5: Each satellite emits a short signal at time t_0

Four landmarks (or satellites) emit a brief signal at the same time t_0 which propagates with a speed c. Each emitted signal contains the identifier of the landmark, its position and the emission time t_0. The robot (which does not have an accurate clock, only an accurate chronometer) receives the four signals at times t_i. From this it easily deduces the offsets between the reception times $\tau_i = t_i - t_1$ (see Figure 2.6). We thus obtain the four equations:

$$\underbrace{\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}}_{\delta_i(x,y,z)} = c(\tau_i + t_1 - t_0),\ i \in \{1,2,3,4\} \tag{2.3}$$

where the parameters, whose values are known with high precision, are c,t_0,x_i, y_i, z_i,τ_i. The four unknowns are x, y, z, t_1. Solving this system allows it to be localized and also to readjust its clock (through t_1). In the case of the GNSS, the landmarks are mobile. They use a similar principle to be localized and synchronized, from fixed landmarks on the ground.

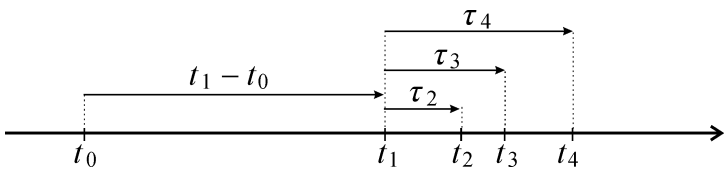


Figure 2.6: The emission time t_0 and the offsets between the arrival times τ_i are all known

Note that, the unknown time t_1 can easily be eliminated by subtraction of the equations 2.3 for

different i. For landmarks i, j, we get

$$\delta_i(x,y,z) - \delta_j(x,y,z) = c(\tau_i - \tau_j).$$

This explains why the technique, called TDOA (Time-difference of arrival), assumes that the difference of the distances $\delta_i - \delta_j$ is measured.

2.4 Full state space model

The full 2D evolution state space model of the DD-Boat is given by

$$\underbrace{\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{\omega} \\ \dot{\omega}_1 \\ \dot{\omega}_2 \end{pmatrix}}_{\dot{\mathbf{x}}} = \underbrace{\begin{pmatrix} v_x \cos\theta - v_y \sin\theta \\ v_x \sin\theta + v_y \cos\theta \\ \omega \\ \omega v_y - p_5 v_x |v_x| + p_3(\omega_1|\omega_1| + \omega_2|\omega_2|) \\ -\omega v_x - p_6 v_y |v_y| \\ -p_7 \omega|\omega| + p_4(\omega_2|\omega_2| - \omega_1|\omega_1|) \\ -p_1\omega_1|\omega_1| + p_2 u_1 \\ -p_1\omega_2|\omega_2| + p_2 u_2 \end{pmatrix}}_{\mathbf{f}(\mathbf{x},\mathbf{u})}$$

where (x, y) in $[m]$ are the coordinates of the robot in the local frame, θ[rad] is the heading, (v_x, v_y) in $[m/sec]$ is the speed vector of the boat in its own frame, ω[rad/sec] is the rotation rate of the boat and ω_1, ω_2[rad/sec] are the speeds of the propellers.

The observation equation is given by

$$\underbrace{\begin{pmatrix} \ell_x \\ \ell_y \\ m_1 \\ m_2 \\ m_3 \\ \omega \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} \ell_x^0 + \frac{\cos\psi_0 x + \sin\psi_0 y}{\rho \cdot \cos \ell_y^0} \\ \ell_y^0 + \frac{-\sin\psi_0 x + \cos\psi_0 y}{\rho} \\ \sin(\theta + \psi_0)\cos I \\ \cos(\theta + \psi_0)\cos I \\ -\sin I \\ \omega \end{pmatrix}}_{\mathbf{g}(\mathbf{x},\mathbf{u})} \tag{2.4}$$

2.5 Observer

Since we do not measure directly the state variables, to implement a control loop, we need an observer. We have no access to the speed of the propellers, neither to the speeds v_x, v_y of the vehicle. Since

$$\begin{pmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{\omega}_1 \\ \dot{\omega}_2 \end{pmatrix} = \begin{pmatrix} \omega v_y - p_5 v_x|v_x| + p_3(\omega_1|\omega_1| + \omega_2|\omega_2|) \\ -\omega v_x - p_6 v_y|v_y| \\ -p_1\omega_1|\omega_1| + p_2 u_1 \\ -p_1\omega_2|\omega_2| + p_2 u_2 \end{pmatrix}$$

is stable, we can use an open loop observer given by

$$\begin{pmatrix} \dot{\hat{v}}_x \\ \dot{\hat{v}}_y \\ \dot{\hat{\omega}}_1 \\ \dot{\hat{\omega}}_2 \end{pmatrix} = \begin{pmatrix} \omega \hat{v}_y - p_5 \hat{v}_x|\hat{v}_x| + p_3(\hat{\omega}_1|\hat{\omega}_1| + \hat{\omega}_2|\hat{\omega}_2|) \\ -\omega \hat{v}_x - p_6 \hat{v}_y|\hat{v}_y| \\ -p_1\hat{\omega}_1|\hat{\omega}_1| + p_2 u_1 \\ -p_1\hat{\omega}_2|\hat{\omega}_2| + p_2 u_2 \end{pmatrix}$$

Moreover, using (2.2) and (2.4), we get the following estimator for x, y, θ, ω:

$$\begin{pmatrix} \hat{x} \\ \hat{y} \\ \hat{\theta} \\ \hat{\omega} \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} \cos\psi_0 & -\sin\psi_0 \\ \sin\psi_0 & \cos\psi_0 \end{pmatrix} \cdot \begin{pmatrix} \rho \cdot \cos \ell_y^0 \cdot (\ell_x - \ell_x^0) \\ \rho \cdot (\ell_y - \ell_y^0) \end{pmatrix} \\ \text{atan2}(m_1, m_2) - \psi_0 \\ \omega \end{pmatrix}$$

Figure 2.7 illustrates the notion of observer to estimate the state vector $\mathbf{x}$ using $\mathbf{u}$ and $\mathbf{y}$ only. The controller can be obtained using a feedback linearisation or a line following strategy as explained in Chapter 1. The parameters p_i have to be identified using some experiments in the pool.

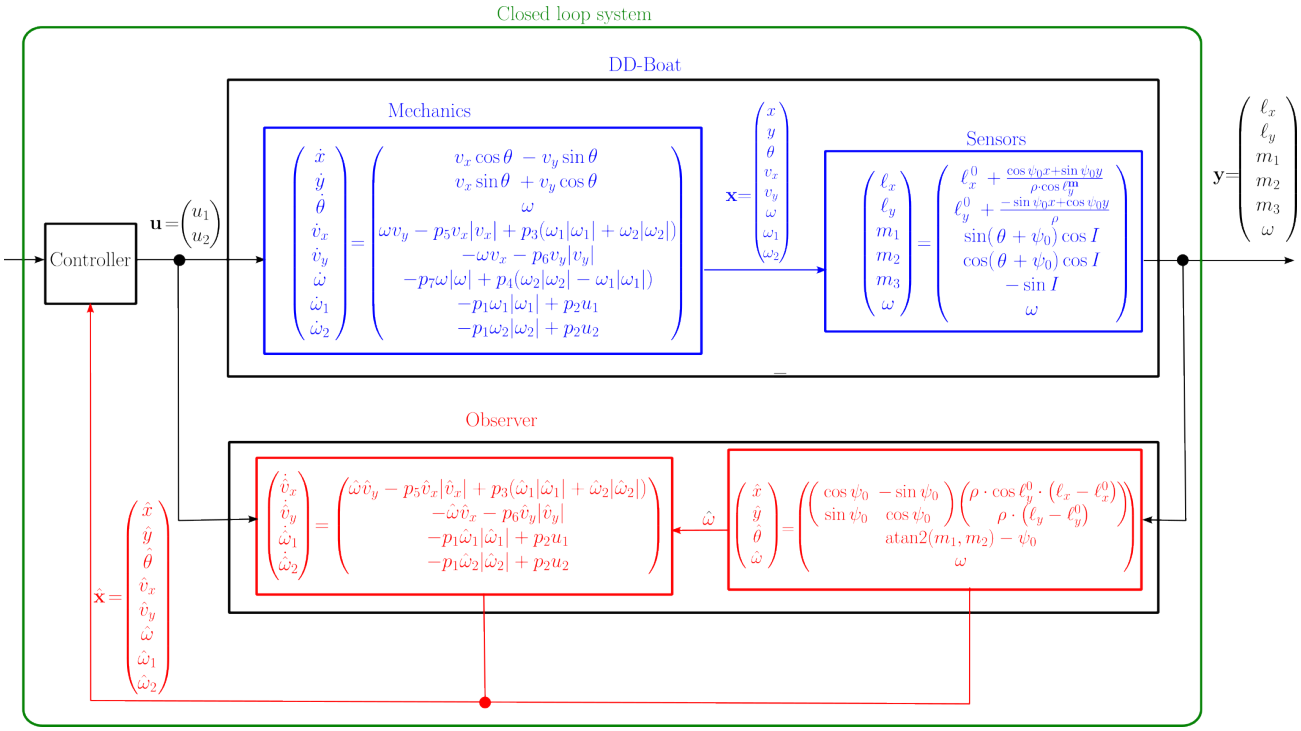


Figure 2.7: Model of the boat (blue) with the observer (red)

Exercises

EXERCISE 4.– *Simulator of a Dubins car in Google Earth*

Consider a robot (a *Dubins car*) described by the following state equations:

$$\begin{cases} \dot{x} &= (u_1 + u_2) \cdot \cos\theta \\ \dot{y} &= (u_1 + u_2) \cdot \sin\theta \\ \dot{\theta} &= u_2 - u_1 \end{cases}$$

where θ is its orientation and (x, y) are the coordinates of its center.

1) Provide a simulation in the pool in local coordinates. The starter code is named `dubins_simulator.py`.

2) Generate the corresponding KML file, taking for the reference longitude $\ell_x^0 = -0.07800405$, reference latitude ℓ_y^0 =0.84507466[rad]. The orientation of the pool is taken as 356.12deg.

3) Draw the corresponding trajectory in the Google Earth. The results should be similar to Figure 2.8.

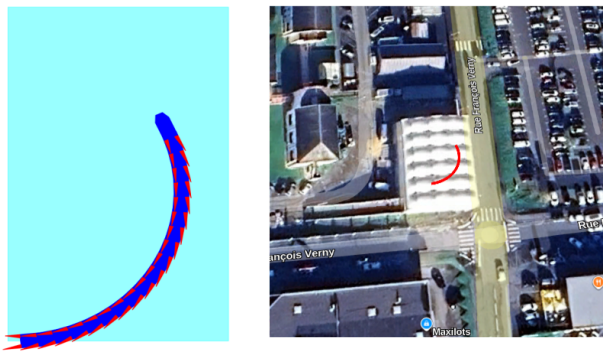


Figure 2.8: Simulation of a Dubins car

EXERCISE 5.– *DD-Boat: Tuning parameters*

The DD-Boat can be described by the following state equations

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{\omega} \\ \dot{\omega}_1 \\ \dot{\omega}_2 \end{pmatrix} = \begin{pmatrix} v_x \cos\theta - v_y \sin\theta \\ v_x \sin\theta + v_y \cos\theta \\ \omega \\ \omega v_y - p_5 v_x |v_x| + p_3(\omega_1|\omega_1| + \omega_2|\omega_2|) \\ -\omega v_x - p_6 v_y |v_y| \\ -p_7 \omega|\omega| + p_4(\omega_2|\omega_2| - \omega_1|\omega_1|) \\ -p_1 \omega_1|\omega_1| + p_2 u_1 \\ -p_1 \omega_2|\omega_2| + p_2 u_2 \end{pmatrix}$$

where the voltages u_1, u_2 both belong to the interval $[-5, 5]$. See the video: https://youtu.be/TsvEUGa-XAs to have an idea of the behavior of the boat. We want to find some realistic values for the parameters p_i using basic experiments and data sheets.

1) When the voltage is $u_i = u_{\max} = 5V$, the corresponding propeller turns at a maximal speed of $\omega_i^{\max} = 3800$RPM (revolution per minute) when underwater. Moreover, if the propeller starts from a standstill, it reaches half of its max speed in $\tau_u = 0.02sec$. Compute p_1, p_2.

2) The maximal acceleration we can obtain for the DD-Boat is $\dot{v}_x = 2\text{ms}^{-2}$. Deduce a value for p_3.

3) The max speed of the boat is $v_x = 5$m/sec. Compute p_5.

4) When we start with a static position, propellers off, but with a rotation rate $\omega(0) = 1$rad/sec, the speed is reduced by $\frac{1}{2}$ after $\tau_\omega = 0.2$sec. Compute p_7.

5) The maximal rotation rate for the boat is $\omega^{\max} = 3$rad/sec. Compute p_4.

6) When we start with a static position, propellers off, but with a lateral speed of 1m/sec $v_y(0) = 1m$/sec, the speed is reduced by $\frac{1}{2}$ after $\tau_y = 0.2$sec. Compute p_6.

7) Validate all these values for the parameters using simulations.

Exercise 6.– *DD-Boat simulator*

The DD-Boat can be described by the following state equations

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{\omega} \\ \dot{\omega}_1 \\ \dot{\omega}_2 \end{pmatrix} = \begin{pmatrix} v_x \cos\theta - v_y \sin\theta \\ v_x \sin\theta + v_y \cos\theta \\ \omega \\ \omega v_y - p_5 v_x |v_x| + p_3(\omega_1|\omega_1| + \omega_2|\omega_2|) \\ -\omega v_x - p_6 v_y |v_y| \\ -p_7 \omega|\omega| + p_4(\omega_2|\omega_2| - \omega_1|\omega_1|) \\ -p_1 \omega_1|\omega_1| + p_2 u_1 \\ -p_1 \omega_2|\omega_2| + p_2 u_2 \end{pmatrix}$$

where the voltages u_1, u_2 both belong to the interval $[-5, 5]$. We assume that the parameters are given by

$$p_1 = 0.07, p_2 = 220, p_3 = 3 \cdot 10^{-5}, p_4 = 15 \cdot 10^{-5}, p_5 = 0.4, p_6 = 5, p_7 = 5$$

1) Provide a simulation on the pool ($12m \times 20m$). The starter code is named `ddboat_simulator.py`.

2) Generate the data $\boldsymbol{\ell} = (\ell_x, \ell_y)$ that should have been generated by the GPS. For the Robotics pool, the reference point is taken as $4°28'9.49"W$ and $48°25'9.16"N$ (see Figure 2.9) which corresponds to:

$$\begin{array}{rcl} \ell_x^0 & = & -0.07800405[\text{rad}] \\ \ell_y^0 & = & 0.84507466[\text{rad}] \end{array}$$

Visualize the trajectory with Google Earth. For this, you need to generate a KML file.

3) Generate the data $\mathbf{y}$ that should have been generated by a calibrated magnetometers. Add some realistic noise.

4) Draw a trajectory generated by your simulator, both in the pool with and in Google Earth.

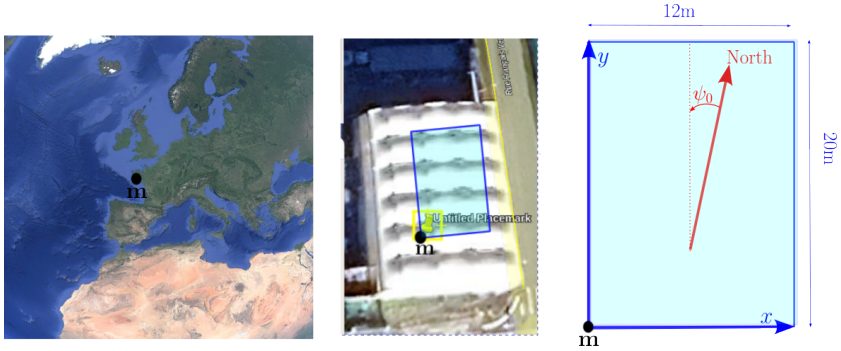


Figure 2.9: Pool represented in Google Earth

EXERCISE 7.– *Navigation-Guidance-Control*

In this exercise, we implement a control loop for the DD-Boat, based on the realistic simulator used in the previous exercise. The global controller we propose is known as NGV (Navigation, Guidance, Control) and is illustrated by Figure 2.10. Navigation determines the vehicle's position, speed, and orientation in the pool frame. Guidance decides the path or direction needed to reach the target, in the pool frame. The feedforward control generates commands to the actuators so the vehicle follows the guided path.

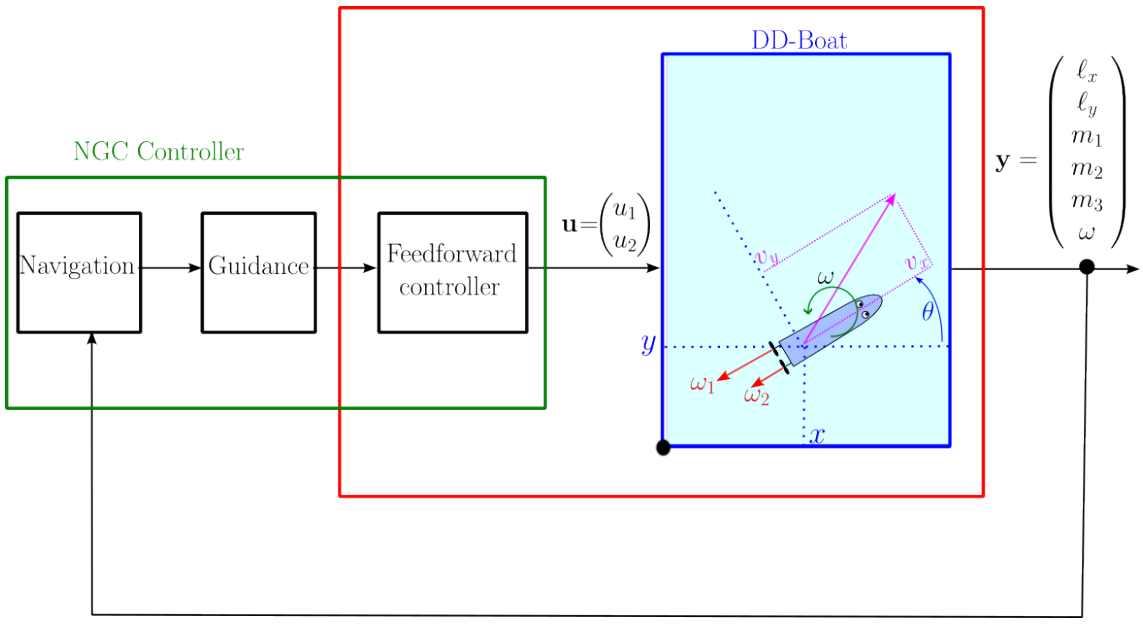


Figure 2.10: NGC controller

1) Provide an openloop feedforward controller to make the DD-Boat behave like a Dubins car, *i.e.*, the red block of the Figure should have approximately the form

$$\begin{cases} \dot{x} &= u_1 \cdot \cos\theta \\ \dot{y} &= u_1 \cdot \sin\theta \\ \dot{\theta} &= u_2 \end{cases}$$

Simulate in Python. The starter code is named `ddboat_NGC.py`.

2) Propose a guidance controller to go to point $\mathbf{a} = (12, 20)$. The inputs of the guidance is the position and orientation of the DD-Boat in the pool frame.

3) Add the navigation block to use only the sensors of the DD-Boat (*i.e.*, a GPS and a magnetometer).

Chapter 3

Magnetism

For thousands of years, long before maps, compasses, or GPS existed, both animals and humans were able to travel across great distances and still find their way. Many living beings rely on natural navigation systems, tools provided by the environment itself. One of the most important is the Sun, which rises in the east and sets in the west, helping to give a sense of direction during the day. At night, the stars act like a map in the sky by some birds. Experimental proofs have been made to prove these claims. For instance, Frieder Sauer, a German biologist, conducted a series of experiments in a planetarium in 1957. He artificially recreated the sky, changing only its orientation, and placed birds in this environment. The birds Garden Warblers (Fauvette grisette) took a different flight path, which appeared to be based on the orientation of the sky.

Some animals have even more surprising abilities. Certain species can sense the Earth's magnetic field, a phenomenon known as *magnetoreception*. This allows animals like birds, sea turtles, and even some insects to travel thousands of kilometers with remarkable accuracy. Some experiments on sea turtle were tested in laboratory where scientists simulated magnetic fields from different parts of the ocean. The turtles swam in directions appropriate for those locations, even though they had never been there. Moreover, researchers attached small magnets to the marine turtles which became disoriented.

This chapter introduces some useful concepts about Earth's magnetism and how it can be used for navigation of the DD-Boat. Magnetometers are compact sensors designed to measure the Earth's magnetic field, which allows the drone to determine its heading relative to magnetic north. Combined with gyroscopes and accelerometers (in an IMU), they help stabilize orientation, correct drift, and enable reliable navigation.

The Earth generates a weak magnetic field (about 25–65 µT depending on location). A magnetometer measures the vector components of this field along three axes. From these components, the drone's heading can be computed.

Most small drones use magnetometers, based on one of the following effects:

- Hall effect (see Figure 3.1): When an electric current (gray) flows through a conductor (yellow)

placed in a magnetic field, the field causes a transverse voltage. This measured voltage is proportional to the magnetic field strength.

- Magnetoresistive effect (dominant technology): The sensor contains tiny resistive elements whose resistance varies depending on the direction and strength of the magnetic field. By measuring these resistance changes, the sensor determines the magnetic field components.

Now, magnetometers are very sensitive to nearby magnetic disturbances: Motors, wires, batteries, and metal structures distort measurements. These effects are known as hard-iron and soft-iron distortions. Calibration mathematically compensates for these disturbances, ensuring that the measured magnetic field corresponds as closely as possible to the Earth's true magnetic field.

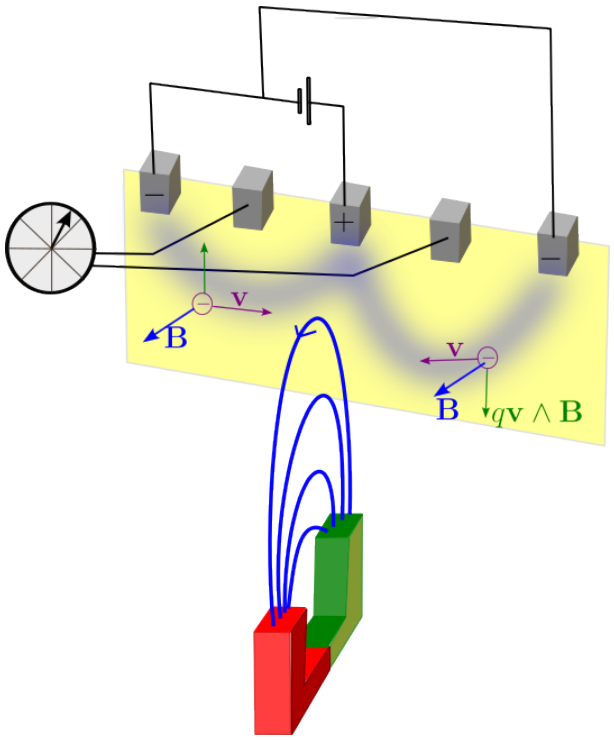


Figure 3.1: Principle of the Hall effect to measure the magnetic field: the trajectories of the electrons are influenced by the magnetic field

3.1 Magnetism on Earth

At large scales, the Earth's magnetic field is well approximated by the field of a *magnetic dipole* $\mathbf{m}$ located at the center of the Earth and aligned (approximately) with the rotation axis (see Figure 3.2). A magnetic dipole can be realized physically by a small current loop carrying a steady current. At a given position $\mathbf{r}$, we have [13]:

$$\mathbf{B}(\mathbf{r}) = \frac{\mu_0}{4\pi\|\mathbf{r}\|^3}\left(3\mathbf{i}_1(\mathbf{m}^{\mathrm{T}}\cdot\mathbf{i}_1) - \mathbf{m}\right) \tag{3.1}$$

where $\mathbf{i}_1 = \mathbf{r}/\|\mathbf{r}\|$ and $\mu_0 = 1.26 \cdot 10^{-6} NA^{-2}$ is the vacuum magnetic permeability. If we are at the surface of the Earth, $\|\mathbf{r}\|$ corresponds to the radius of the Earth.

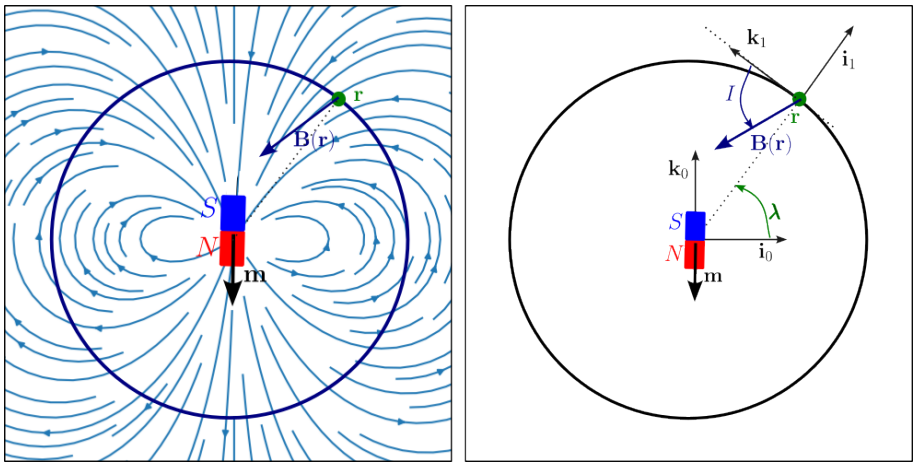


Figure 3.2: Magnetic field at the surface of the Earth

Assume the dipole is aligned with the z-axis, *i.e.*,

$$\mathbf{m} = -m\,\mathbf{k}_0.$$

We have

$$\begin{aligned} \mathbf{B}(\mathbf{r}) &= \frac{\mu_0}{4\pi\|\mathbf{r}\|^3}\left(3\mathbf{i}_1\underbrace{(-m)\sin\lambda}_{\mathbf{m}^{\mathrm{T}}\cdot\mathbf{i}_1} - (-m)\cdot\underbrace{(\sin\lambda\cdot\mathbf{i}_1+\cos\lambda\cdot\mathbf{k}_1)}_{\mathbf{k}_0}\right) \\ &= \frac{\mu_0(-m)}{4\pi\|\mathbf{r}\|^3}\,(2\sin\lambda\mathbf{i}_1 - \cos\lambda\cdot\mathbf{k}_1) \end{aligned}$$

where λ is the latitude. The magnetic inclination I is the angle between $\mathbf{B}(\mathbf{r})$ and the horizontal plane, *i.e.*,

$$\tan I = \frac{2\sin\lambda}{\cos\lambda} = 2\tan\lambda$$

Therefore,

$$I = \arctan(2\tan\lambda)$$

A three dimensional view of the vector field $\mathbf{B}(\mathbf{r})$ is represented by Figure 3.3.

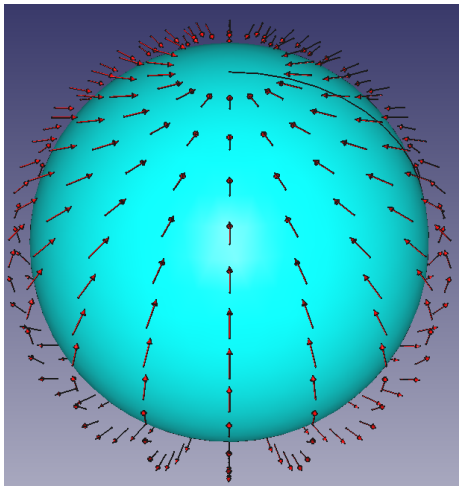


Figure 3.3: Magnetic field at the surface of the Earth

In Brest, the inclination of the magnetic field is: $I = 64^\circ$ and its intensity is : $\beta = 46.0\mu T$. See: franceipgp_Jussieu.html

3.2 Magnetic field distortion

3.2.1 Principle

We consider a uniform magnetic field $\mathbf{B}_0$ applied along the x–axis which can represent the Earth magnetic field in a given area, as for instance, near the pool. Consider a small straight ferromagnetic rod of volume V centered at the origin. Its axis makes an angle θ with the x–axis as illustrated by Figure 3.4.

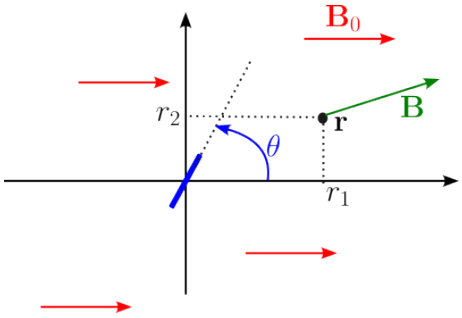


Figure 3.4: Distorsion of a magnetic field by the rod (blue)

The rod is assumed to be linear ferromagnetic (constant susceptibility χ), unsaturated, and small compared to the observation distance. Our goal is to determine the magnetic field $\mathbf{B}(x_1, x_2)$ in the

plane outside the rod. The rod can be modeled as a magnetic dipole. The induced magnetic dipole moment is [13]:

$$\mathbf{m} = \chi V \frac{B_0}{\mu_0} \cos\theta \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} \tag{3.2}$$

The magnetic field generated at point $\mathbf{r}$ by the rod (see (3.1)) is

$$\mathbf{B}_{\text{dip}}(\mathbf{r}) = \frac{\mu_0}{4\pi}\left(\frac{3\mathbf{r}(\mathbf{m}\cdot\mathbf{r})}{r^5} - \frac{\mathbf{m}}{r^3}\right) \tag{3.3}$$

$$\begin{aligned}
\mathbf{B}_{\text{dip}}(\mathbf{r}) &= \frac{\mu_0}{4\pi}\left(\frac{3\left(\chi V \frac{B_0}{\mu_0}\cos\theta \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}\right)\begin{pmatrix} r_1 \\ r_2 \end{pmatrix}\right)\begin{pmatrix} r_1 \\ r_2 \end{pmatrix}}{r^5} - \frac{\chi V \frac{B_0}{\mu_0}\cos\theta \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}}{r^3}\right) \\
&= \frac{\chi V B_0}{4\pi r^5}\cos\theta\left(3\left(x_1\cos\theta + x_2\sin\theta\right)\begin{pmatrix} r_1 \\ r_2 \end{pmatrix} - r^2\begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}\right) \\
&= \frac{\chi V B_0}{4\pi r^5}\cos\theta\begin{pmatrix} 3r_1(r_1\cos\theta + r_2\sin\theta) - r^2\cos\theta \\ 3r_2(r_1\cos\theta + r_2\sin\theta) - r^2\sin\theta \end{pmatrix}
\end{aligned}$$

with

$$\mathbf{r} = (r_1, r_2), \quad r = \|\mathbf{r}\|.$$

The total magnetic field is therefore

$$\mathbf{B}(\mathbf{r}) = \mathbf{B}_0 + \mathbf{B}_{\text{dip}}(\mathbf{r})$$

Note that the approximation is valid far enough from the rod and in the linear magnetization regime. An illustration is given by Figure 3.5.

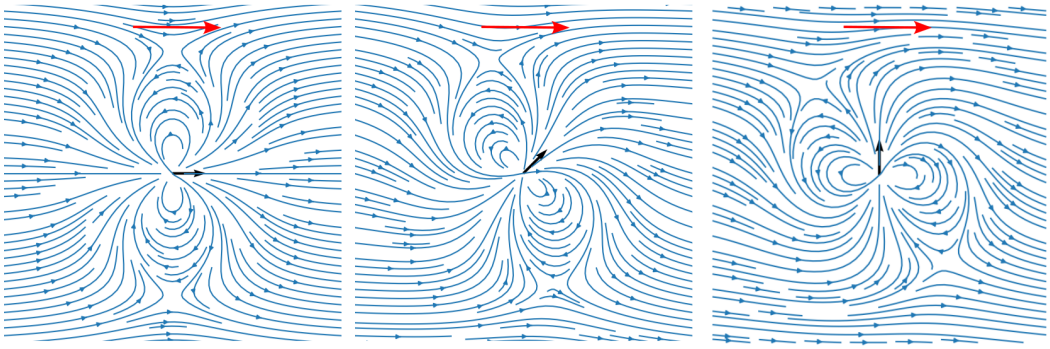


Figure 3.5: Distortion of a magnetic field, by a ferromagnetic rod

3.2.2 Table experiment

Take a small straight ferromagnetic rod placed on a rotating wooden table (see Figure 3.6). A magnetometer is also fixed on the table at position $\mathbf{r}$ and measures the horizontal magnetic field components (m_x, m_y) in the coordinate system attached to the table. A constant horizontal magnetic field $\mathbf{B}_0$ exists in the room. This field is fixed in the laboratory frame. When the table rotates by an angle γ, the magnetometer frame rotates with the table. Because the magnetometer rotates with the table, the ambient magnetic field appears to rotate in the table frame.

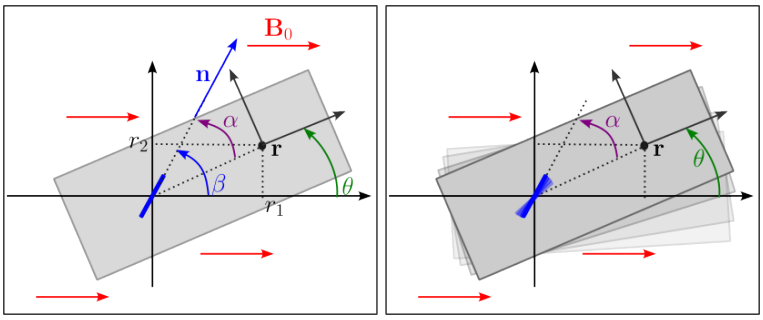


Figure 3.6: A table with, a rod (blue) and the magnetometer in $\mathbf{r}$

Without the rod, the magnetometer would measure

$$\begin{pmatrix} m_x \\ m_y \end{pmatrix} = \begin{pmatrix} B_0 \cos\theta \\ -B_0 \sin\theta \end{pmatrix}$$

which describes a circle in the (m_x, m_y) plane, when the table rotates. Put a soft ferromagnetic rod fixed to the table (*i.e.,* α remains constant) which becomes magnetized mainly along its axis. Using (3.2), we know that:

$$\mathbf{m} \propto \cos\beta\, \mathbf{n} \tag{3.4}$$

where $\mathbf{n}$ is the unit vector for the orientation of the rod. The relation (3.3), given by,

$$\mathbf{B}_{\text{dip}}(\mathbf{r}) \propto \frac{3\mathbf{r}(\mathbf{m}^{\mathrm{T}}\mathbf{r})}{r^5} - \frac{\mathbf{m}}{r^3}$$

becomes

$$\mathbf{B}_{\text{dip}}(\mathbf{r}) \propto \cos\beta \cdot \underbrace{\left(\frac{3\mathbf{r}(\mathbf{n}^{\mathrm{T}}\mathbf{r})}{r^5} - \frac{\mathbf{n}}{r^3} \right)}_{\mathbf{w}} = \cos(\theta+\alpha) \cdot \mathbf{w}$$

The vector $\mathbf{w}$ is constant is the table frame. In the table frame, we get that the measured magnetic field is

$$\begin{pmatrix} m_x \\ m_y \end{pmatrix} = \mathbf{B}_{\text{dip}} + \begin{pmatrix} B_0 \cos\theta \\ -B_0 \sin\theta \end{pmatrix}$$

Since $\cos(\theta+\alpha) = \cos\theta \cdot \cos\alpha - \sin\theta \cdot \sin\alpha$, we get that m_x and m_y are both linear in $\cos\theta$ and $\sin\theta$, *i.e.*,

$$\begin{pmatrix} m_x \\ m_y \end{pmatrix} = \mathbf{A} \cdot \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}.$$

which describes a centered ellipse in the (m_x, m_y)-plane, when the table rotates. With n rods, a similar reasoning would yield a similar result. The conclusion of this experiment is that if we measure magnetic field with distortion of a magnetic field due to soft iron, then, in the sensor frame, whereas the table (or the DD-Boat) is rotating, we always get a centered ellipse (or an ellipsoid in 3D).

3.2.3 Inside the DD-Boat

Fix a magnetometer inside the DD-Boat. The sensor does not measure only the Earth's magnetic field. It also "sees" magnetic effects created by the robot itself. To understand this, we can imagine that there are many tiny magnets inside the robot. When the boat moves, some of these tiny magnets are free to turn to be aligned with the Earth's magnetic field. They do not have a fixed direction, but they are easily influenced by the surrounding field. These corresponds to *soft iron* and the result corresponds to the magnetic distortion introduced in the previous section. Soft-iron effects distort the Earth's magnetic field: they stretch it or bend it, but they do not create a constant magnetic field on their own. Their effect depends on the robot's orientation.

Other tiny magnets are fixed to the robot. They keep the same direction in the robot's frame no matter how it moves. These correspond to *hard iron*. Hard-iron effects add a constant magnetic field to the measurement, shifting the magnetometer readings by a fixed amount.

During magnetometer calibration, we try to identify and remove both effects: hard iron causes a shift of the measurements, soft iron causes a deformation of the measurements (see Figure 3.7, bottom). By compensating both, the magnetometer can correctly measure the Earth's magnetic field and give an accurate heading.

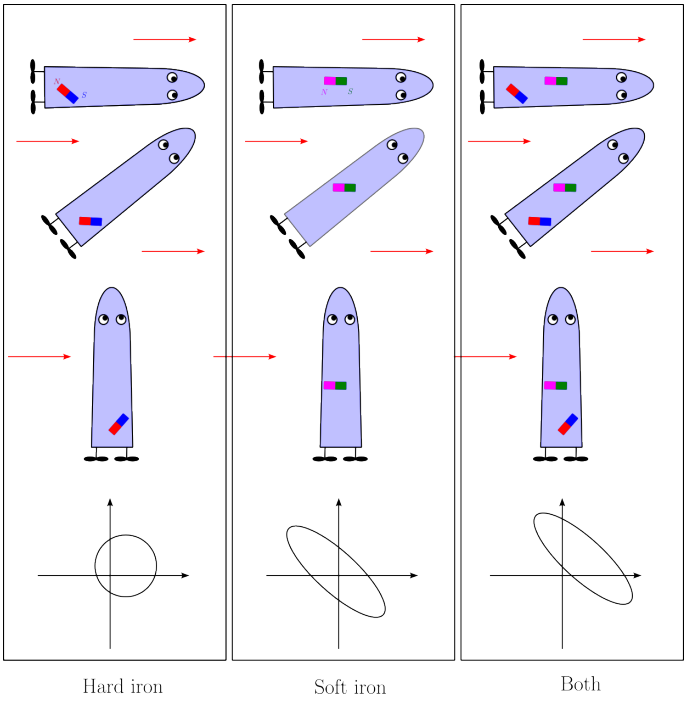


Figure 3.7: Left : hard iron perturbation, Center: solf iron; Right: Both

3.3 Calibration of the magnetometers

When the magnetic field is measured in the DD-Boat, two types of perturbation occur

- Soft iron effects are caused by nearby materials that distort the Earth's magnetic field. They don't create their own magnetic field, but they bend and reshape the field lines depending on orientation. This results in direction-dependent scaling and skewing of compass measurements (often seen as an ellipse instead of a circle when rotating the sensor).
- Hard iron effects come from permanently magnetized objects near the compass (like magnets, speakers, or magnetized screws). These objects add a constant magnetic offset to the Earth's field, shifting all compass readings in one direction regardless of orientation.

In calibration terms, hard iron is corrected by offset (bias) removal and soft iron is corrected by scaling and axis alignment (matrix correction).

Denote by $\mathbf{y} \in \mathbb{R}^3$ the magnetic field that exists at the robot's position if we remove the robot. It the correct magnetic field, *i.e.*, the one we would like to measure. Unfortunately, the raw magnetometer returns a vector of magnetic measurements $\mathbf{x} \in \mathbb{R}^3$ that is corrupted by the presence of the robot. We can assume that $\mathbf{x}$ and $\mathbf{y}$ are linked by the linear equation:

$$\mathbf{x} = \mathbf{A}\mathbf{y} + \bar{\mathbf{x}}.$$

The calibration step searches for an estimation $\hat{\mathbf{A}}$ and $\hat{\bar{\mathbf{x}}}$ for $\mathbf{A}$ and $\bar{\mathbf{x}}$. It is obtained by a sequence of measurement made at a given position of the robot, but for different orientations, as illustrated by Figure 3.8. These quantities depend on the boat only.

The *corrector* is given by

$$\hat{\mathbf{y}} = \hat{\mathbf{A}}^{-1}(\mathbf{x} - \hat{\bar{\mathbf{x}}})$$

The *calibrated magnetometer* includes both the raw magnetometer and the corrector.

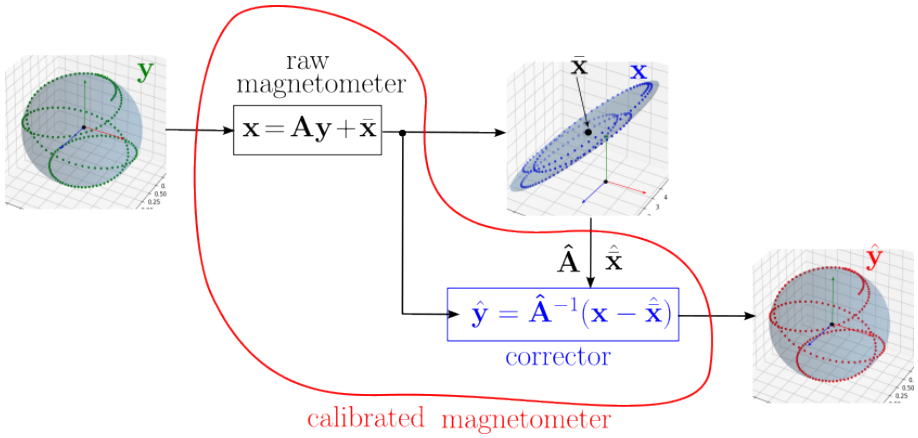


Figure 3.8: Calibrated magnetometer

3.4 Magnetometer autocalibration

Magnetometer autocalibration is the process of determining hard- and soft-iron calibration parameters from arbitrary 3D orientation data, without requiring known attitude or external references. Equivalently, we want to find a transformation that converts the distorted ellipsoid into a sphere centered at the origin, to correct for hard- and soft-iron effects, without taking care of the rotation part which can be known in advance.

3.4.1 Symmetric positive definite matrix

We recall first some basic notions on symmetric positive definite matrices that will be used in this section. The matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is called *symmetric positive definite* (we write $\mathbf{A} \succ 0$) if

$$\begin{array}{l} \mathbf{A}^{\mathrm{T}} = \mathbf{A} \\ \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} > 0 \quad , \quad \forall \mathbf{x} \in \mathbb{R}^n, \ \mathbf{x} \neq 0 \end{array}$$

The set of all $\mathbf{A} \in \mathbb{R}^{n \times n}$ that are symmetric positive definite is denoted by $S^+(n)$.

A consequence of this definition is that the eigenvalues of $\mathbf{A} \succ 0$ are strictly positive.

Diagonalization. Every $\mathbf{A} \in S^+(n)$ admits an orthogonal diagonalization, i.e., there exists an orthogonal matrix $\mathbf{R}$ and a diagonal matrix

$$\mathbf{D} = \text{diag}(\lambda_1, \ldots, \lambda_n)$$

such that

$$\mathbf{A} = \mathbf{R}\mathbf{D}\mathbf{R}^{\mathrm{T}}.$$

This decomposition explains most algebraic properties of positive definite matrices.

Positive cone. The set of $S^+(n)$ is a positive cone, *i.e.*, $S^+(n)$ is closed under addition and multiplication by $\alpha > 0$.

Square. If $\mathbf{A} \in S^+(n)$ then $\mathbf{A}^2 \in S^+(n)$. Indeed, using the spectral decomposition, since

$$\mathbf{A} = \mathbf{R}\mathbf{D}\mathbf{R}^{\mathrm{T}}$$

we obtain

$$\mathbf{A}^2 = \mathbf{R}\mathbf{D}\mathbf{R}^{\mathrm{T}}\mathbf{R}\mathbf{D}\mathbf{R}^{\mathrm{T}} = \mathbf{R}\mathbf{D}^2\mathbf{R}^{\mathrm{T}}$$

so the eigenvalues become λ_i^2, which remain strictly positive.

Matrix Functions. Because the eigenvalues of $\mathbf{A} \in S^+(n)$ are positive, many functions can be applied to the matrix through its spectral decomposition. If $\mathbf{A} = \mathbf{R}\mathbf{D}\mathbf{R}^{\mathrm{T}}$, and f is a function defined on positive numbers, one can define

$$f(\mathbf{A}) = \mathbf{R} \begin{pmatrix} f(\lambda_1) & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & f(\lambda_n) \end{pmatrix} \mathbf{R}^{\mathrm{T}}.$$

This allows the definition of matrices such as $\sqrt{\mathbf{A}}, \mathbf{A}^{-1}, \mathbf{A}^{\alpha}, \log(\mathbf{A}), \exp(\mathbf{A})$.

3.4.2 Fit the ellipsoid

We have many points $(x_1(i), x_2(i), x_3(i))$, $i \geq 1$, collected by the magnetometer. All belong to an ellipsoid:

$$\underbrace{p_1x_1^2 + p_2x_2^2 + p_3x_3^2 + p_4x_1x_2 + p_5x_1x_3 + p_6x_2x_3 + p_7x_1 + p_8x_2 + p_9x_3}_{\mathbf{f_p(x)}} = 1 \tag{3.5}$$

i.e.,

$$\left(x_1^2|x_2^2|x_3^2|x_1x_2|x_1x_3|x_2x_3|x_1|x_2|x_3\right) \cdot \mathbf{p} = 1$$

We should have

$$\underbrace{\begin{pmatrix} x_1^2(1) & x_2^2(1) & x_3^2(1) & x_1(1)x_2(1) & x_1(1)x_3(1) & x_2(1)x_3(1) & x_1(1) & x_2(1) & x_3(1) \\ x_1^2(2) & x_2^2(2) & x_3^2(2) & x_1(2)x_2(2) & x_1(2)x_3(2) & x_2(2)x_3(2) & x_1(2) & x_2(2) & x_3(2) \\ x_1^2(3) & x_2^2(3) & x_3^2(3) & x_1(3)x_2(3) & x_1(3)x_3(3) & x_2(3)x_3(3) & x_1(3) & x_2(3) & x_3(3) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}}_{\mathbf{X}} \cdot \mathbf{p} = \underbrace{\begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \end{pmatrix}}_{\mathbf{1}} \tag{3.6}$$

A least-square solution of (3.6) is

$$\mathbf{p} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1} \cdot \mathbf{X}^{\mathrm{T}} \cdot \mathbf{1}$$

Proposition 2. *Consider the general quadratic surface in* $\mathbb{R}^3$*:*

$$p_1x_1^2 + p_2x_2^2 + p_3x_3^2 + p_4x_1x_2 + p_5x_1x_3 + p_6x_2x_3 + p_7x_1 + p_8x_2 + p_9x_3 = 1$$

or equivalently

$$\mathbf{x}^T \underbrace{\begin{pmatrix} p_1 & \frac{p_4}{2} & \frac{p_5}{2} \\ \frac{p_4}{2} & p_2 & \frac{p_6}{2} \\ \frac{p_5}{2} & \frac{p_6}{2} & p_3 \end{pmatrix}}_{\mathbf{P}} \mathbf{x} + \underbrace{\mathbf{c}^T}_{(p_7,p_8,p_9)} \cdot \mathbf{x} = 1$$

If the surface corresponds to an ellipsoid, we can write the equation in the canonical quadratic (matrix) form :

$$(\mathbf{x} - \bar{\mathbf{x}})^T \mathbf{Q} (\mathbf{x} - \bar{\mathbf{x}}) = 1$$

where

$$\begin{array}{rcl} \bar{\mathbf{x}} & = & -\frac{1}{2}\mathbf{P}^{-1}\mathbf{c} \\ \mathbf{Q} & = & \frac{\mathbf{P}}{1+\bar{\mathbf{x}}^T\mathbf{P}\bar{\mathbf{x}}} \end{array}$$

The vector $\bar{\mathbf{x}} \in \mathbb{R}^3$ *is the center of the ellipsoid and* $\mathbf{Q}$ *is a symmetric positive definite matrix.*

Proof. The center $\bar{\mathbf{x}}$ satisfies the stationary condition

$$\frac{d}{d\mathbf{x}} \left(\mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{x} + \mathbf{c}^{\mathrm{T}}\mathbf{x}\right) = \mathbf{0}$$

i.e.,

$$2\mathbf{P}\bar{\mathbf{x}} + \mathbf{c} = \mathbf{0}$$

Thus,

$$\bar{\mathbf{x}} = -\tfrac{1}{2}\mathbf{P}^{-1}\mathbf{c}.$$

Set $\mathbf{z} = \mathbf{x} - \bar{\mathbf{x}}$ to shift the coordinate system, as illustrated by Figure 3.9. We have

$$(\mathbf{z} + \bar{\mathbf{x}})^T \mathbf{P} (\mathbf{z} + \bar{\mathbf{x}}) + \mathbf{c}^T (\mathbf{z} + \bar{\mathbf{x}}) = 1$$

i.e.,

$$\mathbf{z}^T\mathbf{P}\mathbf{z} + \underbrace{\bar{\mathbf{x}}^{\mathrm{T}}\mathbf{P}\mathbf{z} + \mathbf{z}^{\mathrm{T}}\mathbf{P}\bar{\mathbf{x}} + \mathbf{c}^{\mathrm{T}}\mathbf{z}}_{=\mathbf{0}(\text{no linear terms})} + \bar{\mathbf{x}}^{\mathrm{T}}\mathbf{P}\bar{\mathbf{x}} + \mathbf{c}^{\mathrm{T}}\bar{\mathbf{x}} = 1$$

The fact that the term linear in $\mathbf{z}$, in the previous equations, cancels out is due to the fact that the minimum is reached for $\mathbf{z} = \mathbf{0}$. Since $\mathbf{c} = -2\mathbf{P}\bar{\mathbf{x}}$, we get

$$\mathbf{z}^T\mathbf{P}\mathbf{z} - \bar{\mathbf{x}}^{\mathrm{T}}\mathbf{P}\bar{\mathbf{x}} = 1$$

i.e.,

$$\mathbf{z}^T\mathbf{P}\mathbf{z} = 1 + \bar{\mathbf{x}}^T\mathbf{P}\bar{\mathbf{x}}$$

i.e.,

$$\mathbf{z}^T\left(\frac{\mathbf{P}}{1 + \bar{\mathbf{x}}^T\mathbf{P}\bar{\mathbf{x}}}\right)\mathbf{z} = 1$$

Therefore,

$$\mathbf{Q} = \frac{\mathbf{P}}{1 + \bar{\mathbf{x}}^T\mathbf{P}\bar{\mathbf{x}}}$$

which concludes the proof. □

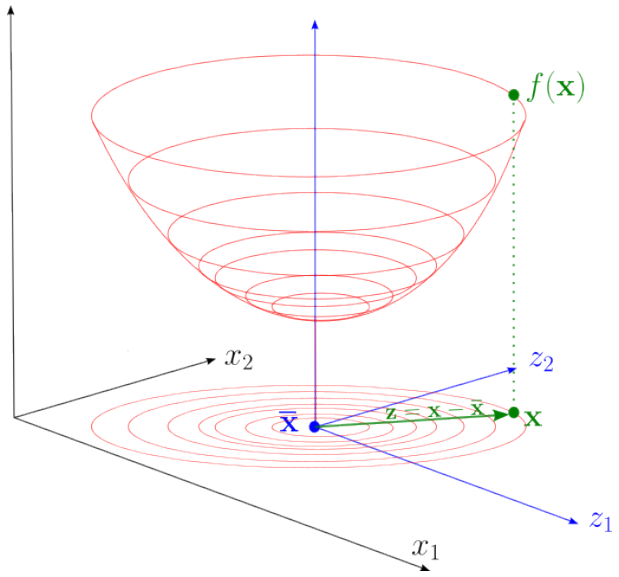


Figure 3.9: Quadratic function

The algorithm to fit the ellipsoid is thus as follows

Algorithm Fit_ellipsoid(In: $\mathbf{x}(i)$; Out :$\mathbf{Q}$,$\bar{\mathbf{x}}$)

1 $\mathbf{X} = \begin{pmatrix} x_1^2(1) & x_2^2(1) & x_3^2(1) & x_1(1)x_2(1) & x_1(1)x_3(1) & x_2(1)x_3(1) & x_1(1) & x_2(1) & x_3(1) \\ x_1^2(2) & x_2^2(2) & x_3^2(2) & x_1(2)x_2(2) & x_1(2)x_3(2) & x_2(2)x_3(2) & x_1(2) & x_2(2) & x_3(2) \\ x_1^2(3) & x_2^2(3) & x_3^2(3) & x_1(3)x_2(3) & x_1(3)x_3(3) & x_2(3)x_3(3) & x_1(3) & x_2(3) & x_3(3) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$

2 $\mathbf{p} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1} \cdot \mathbf{X}^{\mathrm{T}} \cdot \mathbf{1}$

3 $\mathbf{P} = \begin{pmatrix} p_1 & \frac{p_4}{2} & \frac{p_5}{2} \\ \frac{p_4}{2} & p_2 & \frac{p_6}{2} \\ \frac{p_5}{2} & \frac{p_6}{2} & p_3 \end{pmatrix}; \mathbf{c} = \begin{pmatrix} p_7 \\ p_8 \\ p_9 \end{pmatrix}$

4 $\bar{\mathbf{x}} = -\frac{1}{2}\mathbf{P}^{-1}\mathbf{c}$; $\mathbf{Q} = \frac{\mathbf{P}}{1+\bar{\mathbf{x}}^T\mathbf{P}\bar{\mathbf{x}}}$

5 Return $\mathbf{Q}, \bar{\mathbf{x}}$

The corresponding Python code is the following

```
def Fit_Ellipsoid(x1,x2,x3):
   x1,x2,x3=np.array(x1),np.array(x2),np.array(x3)
   X = np.column_stack([x1**2,x2**2,x3**2,x1*x2,x1*x3,x2*x3,x1,x2,x3])
   p = inv(X.T@X)@X.T @ np.ones((len(x1),1))
   p=p.flatten()
   P = np.array([[   p[0]   ,   1/2*p[3] , 1/2*p[4]],
                 [ 1/2*p[3],     p[1]    , 1/2*p[5]],
                 [ 1/2*p[4],   1/2*p[5] ,  p[2]   ]])
   c=np.array([[p[6]],[p[7]],[p[8]]])
   xbar=-1/2*inv(P)@c
   Q = P/(1+xbar.T@P@xbar)
   return Q,xbar
```

3.4.3 Spherization function

Proposition 3. *The spherization function defined by*

$$\begin{array}{ccc} \mathbb{R}^n & \to & \mathbb{R}^n \\ \mathbf{x} & \mapsto & \mathbf{y} = \sqrt{\mathbf{Q}} \cdot (\mathbf{x} - \bar{\mathbf{x}}) \end{array}$$

reshapes the ellipsoid

$$(\mathbf{x} - \bar{\mathbf{x}})^T \mathbf{Q} (\mathbf{x} - \bar{\mathbf{x}}) = 1$$

into the unit sphere,

Proof. We have

$$\begin{array}{rcc} \mathbf{y}^{\mathrm{T}}\mathbf{y} & = & (\mathbf{x}-\bar{\mathbf{x}})^{\mathrm{T}} \cdot \sqrt{\mathbf{Q}}^{\mathrm{T}} \sqrt{\mathbf{Q}} \cdot (\mathbf{x}-\bar{\mathbf{x}}) \\ & = & (\mathbf{x}-\bar{\mathbf{x}})^{\mathrm{T}} \cdot \mathbf{Q} \cdot (\mathbf{x}-\bar{\mathbf{x}}) \\ & = & 1 \end{array}$$

□

The application

$$\mathbf{x} \mapsto \sqrt{\mathbf{Q}} \cdot (\mathbf{x}-\bar{\mathbf{x}})$$

When we perform an ellipsoid fitting to build the spherization function, it is difficult to verify that we get the correct results. The following points could be checked.

- The cloud $\mathbf{x}(i)$ should belong to a common ellipsoid. Otherwise, probably measurements have been collected in area perturbed with soft iron (a room), a phone in your pocket.
- The cloud $\mathbf{y}(i) = \sqrt{\mathbf{Q}} \cdot (\mathbf{x}(i) - \bar{\mathbf{x}})$ should belong to the unit sphere. Equivalently, the norm of the $\mathbf{y}(i)$ should be equal to 1, approximately.
- The matrix $\mathbf{Q}$ should be symmetric and its eigen values should be real and strictly positive.
- By plotting the cloud $\mathbf{x}(i)$, we may observe that the cloud can indeed be fitted by an ellipsoid but the cloud remains confined to a limited region of that ellipsoid. This indicates that the motion used during calibration is not sufficiently rich. For example, rotating the boat only around the direction of the magnetic field produces essentially a single point on the ellipsoid. To obtain larger elliptical trajectories and better coverage, the sensor should be rotated about an axis that is orthogonal to the magnetic field.

3.4.4 Pointing procedure

As said previously, magnetometer autocalibration requires no known attitude or external references. It finds the transformation to reshape an ellipsoid described by a cloud of points into a unit sphere.

Now, in practice, we may not know exactly how the magnetometer has been positioned inside the boat and a precise pointing procedure should be performed. This means that a rotation should be added to the transformation to rotate the sphere properly. The complete correction should have the form $\hat{\mathbf{y}} = \beta \cdot \mathbf{R}.\sqrt{\mathbf{Q}} \cdot (\mathbf{x} - \bar{\mathbf{x}})$, where β is the magnitude of the magnetic field. For simplicity, we compute the normalized version of this correction:

$$\hat{\mathbf{y}} = \mathbf{R}.\sqrt{\mathbf{Q}} \cdot (\mathbf{x} - \bar{\mathbf{x}})$$

The reason for this is that only the direction of the magnetic field is used to find the heading. Moreover the matrix $\mathbf{A}$ can only be identified modulo the scale factor β. This means that $\beta \cdot \mathbf{A}$ can be identified, but not $\mathbf{A}$ and β separately.

Let us take two measurements $\mathbf{x}_N, \mathbf{x}_W$, with the DD-Boat, as illustrated by Figure 3.10.

- Direction N : The robot is laid flat toward the North
- Direction W : The robot is laid flat toward the West

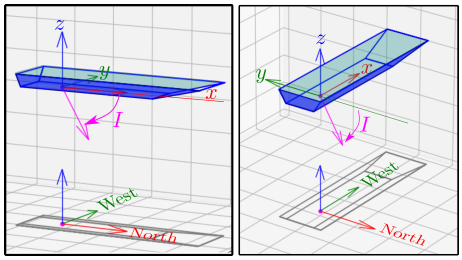


Figure 3.10: Pointing toward two cardinal directions (North and West) to find the rotation correction

The corrected normalized magnetic vector should be

$$\mathbf{y}_N = \begin{pmatrix} \cos I \\ 0 \\ -\sin I \end{pmatrix}, \quad \mathbf{y}_W = \begin{pmatrix} 0 \\ -\cos I \\ -\sin I \end{pmatrix}$$

We have

$$\begin{aligned} \mathbf{y}_N &= \mathbf{R}.\underbrace{\sqrt{\mathbf{Q}} \cdot (\mathbf{x}_N - \bar{\mathbf{x}})}_{\mathbf{v}_N} \\ \mathbf{y}_W &= \mathbf{R}.\underbrace{\sqrt{\mathbf{Q}} \cdot (\mathbf{x}_W - \bar{\mathbf{x}})}_{\mathbf{v}_W} \end{aligned}$$

The cross product of the two previous equations yields:

$$\mathbf{y}_N \wedge \mathbf{y}_W = \mathbf{R}.(\mathbf{v}_N \wedge \mathbf{v}_W)$$

We get

$$\underbrace{\left(\mathbf{y}_N \mid \mathbf{y}_W \mid \mathbf{y}_N \wedge \mathbf{y}_W \right)}_{\mathbf{Y}} = \mathbf{R}.\underbrace{\left(\mathbf{v}_N \mid \mathbf{v}_W \mid \mathbf{v}_N \wedge \mathbf{v}_W \right)}_{\mathbf{V}}$$

Thus

$$\mathbf{R} = \mathbf{Y} \cdot \mathbf{V}^{-1}$$

If $\mathbf{R}$ is not exactly a rotation matrix, we can find the best rotation matrix close to $\mathbf{R}$ using a QR-factorization. The code is given below

```
def projSO3(M): # return a rotation matrix close to M
   Q,R = np.linalg.qr(M)
   return Q@diag(diag(sign(R)))
```

Algorithm MagCalib(In: $\mathbf{x}(i)$,$\mathbf{x}_N, \mathbf{x}_W$; Out :$\hat{\mathbf{A}}^{-1}$,$\bar{\mathbf{x}}$)

$\mathbf{Q}$,$\bar{\mathbf{x}} = \text{Fit\_ellipsoid}(\text{In: } \mathbf{x}(i))$
$I = 64 \cdot \frac{\pi}{180}$;
$\mathbf{v}_N = \sqrt{\mathbf{Q}} \cdot (\mathbf{x}_N - \bar{\mathbf{x}})$
$\mathbf{v}_W = \sqrt{\mathbf{Q}} \cdot (\mathbf{x}_W - \bar{\mathbf{x}})$
$\mathbf{y}_N = \begin{pmatrix} \cos I \\ 0 \\ -\sin I \end{pmatrix}$;$\mathbf{y}_W = \begin{pmatrix} 0 \\ -\cos I \\ -\sin I \end{pmatrix}$
$\mathbf{V} = \begin{pmatrix} \frac{\mathbf{v}_N}{\|\mathbf{v}_N\|} & | & \frac{\mathbf{v}_W}{\|\mathbf{v}_W\|} & | & \frac{\mathbf{v}_N \wedge \mathbf{v}_W}{\|\mathbf{v}_N \wedge \mathbf{v}_W\|} \end{pmatrix}$
$\mathbf{Y} = \begin{pmatrix} \mathbf{y}_N & | & \mathbf{y}_W & | & \mathbf{y}_N \wedge \mathbf{y}_W \end{pmatrix}$
$\mathbf{R} = \mathbf{Y} \cdot \mathbf{V}^{-1}$
$\hat{\mathbf{A}}^{-1} = \mathbf{R}.\sqrt{\mathbf{Q}}$, $\hat{\bar{\mathbf{x}}} = \bar{\mathbf{x}}$
Return $\hat{\mathbf{A}}^{-1}, \hat{\bar{\mathbf{x}}}$

The calibration gives us the corrector

$$\hat{\mathbf{y}} = \hat{\mathbf{A}}^{-1} \cdot (\mathbf{x} - \hat{\bar{\mathbf{x}}})$$

.

The following Python code computes $\hat{\mathbf{A}}^{-1}, \hat{\bar{\mathbf{x}}}$.

```
def mag_calib(x1,x2,x3,xnorth,xwest):
   Q,xbar=Fit_Ellipsoid(x1,x2,x3)
   I=64*3.14/180
   ynorth=np.array([[np.cos(I)],[0],[-np.sin(I)]])
   ywest=np.array([[0],[-np.cos(I)],[-np.sin(I)]])
   yup=np.cross(ynorth.flatten(), ywest.flatten()).reshape(3,1)
   Y=np.hstack([ynorth,ywest,yup])
   vnorth=sqrtm(Q)@(xnorth-xbar)
   vwest=sqrtm(Q)@(xwest-xbar)
   vup=np.cross(vnorth.flatten(),vwest.flatten()).reshape(3,1)
   V=np.hstack([vnorth,vwest,vup])
   R=Y@inv(V)
   invA=R@sqrtm(Q)
   return invA,xbar
```

When we perform the pointing procedure, the following points could be checked.

- The two vectors $\mathbf{x}_N, \mathbf{x}_W$ (`xnorth` and `xwest`) should belong to the ellipsoid made by the $\mathbf{x}(i)$.
- The matrix $\mathbf{R}$ should be (almost) a rotation matrix, which means that $\mathbf{R}\mathbf{R}^{\mathrm{T}} \simeq \mathbf{I}$.

- After calibration, when the magnetometer's x-axis is aligned with the magnetic field, the corrected vector $\hat{\mathbf{y}} = \hat{\mathbf{A}}^{-1} \cdot (\mathbf{x} - \hat{\mathbf{x}})$ should be collinear with the x-axis. Similarly, aligning the field with the y- or z-axis should yield a $\hat{\mathbf{y}}$ collinear with the corresponding axis.

3.5 Great ellipses

Take an ellipsoid $\mathcal{E}$. A great ellipse of $\mathcal{E}$ separates $\mathcal{E}$ into two parts with the same volume. During the calibration, you can perform an great ellipse by rotating the DD-Boat with respect to a vector $\boldsymbol{\omega}$ orthogonal to the magnetic field $\mathbf{B}$, as illustrated by Figure 3.11. Great ellipses generates rich data for an accurate calibration. On the other hand, if you rotate the DD-Boat around a vector $\boldsymbol{\omega}$ collinear to $\mathbf{B}$, you will generate a single point of the ellipsoid, which gives poor information to the calibration process.

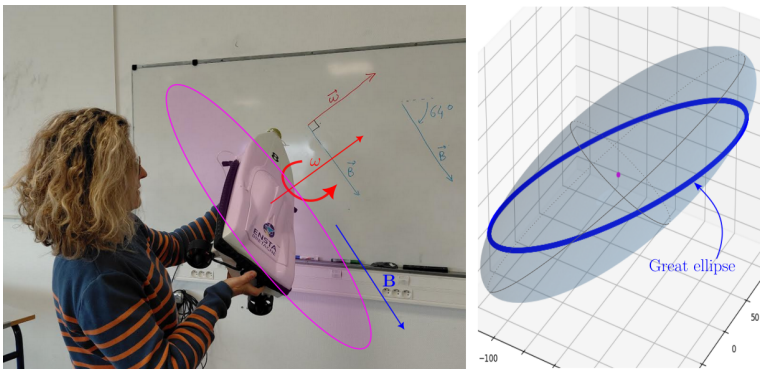


Figure 3.11: Rotating around a vector $\boldsymbol{\omega}$ orthogonal to the magnetic field $\mathbf{B}$, generates a great ellipse

3.6 Detecting walls

A calibrated magnetometer measures the local magnetic field vector with high precision, often at the level of microteslas. It can be expressed in a normalized form as done previously. In an ideal, open environment this field is dominated by the Earth's magnetic field, which is smooth, slowly varying in space, and approximately uniform over human-scale distances. However, when the DD-Boat approaches a wall or other man-made structure, measurable changes appear in the amplitude and direction of the magnetic field. These changes are not measurement artifacts; rather, they arise from well-understood physical interactions between magnetic fields and matter.

The magnetic field measured by a calibrated magnetometer can be written as

$$\hat{\mathbf{y}} = \mathbf{B}_{\text{Earth}} + \mathbf{B}_{\text{induced}} + \mathbf{B}_{\text{remanent}},$$

where $\mathbf{B}_{\text{Earth}}$ is the ambient geomagnetic field, $\mathbf{B}_{\text{induced}}$ arises from magnetization induced in nearby materials, and $\mathbf{B}_{\text{remanent}}$ is due to permanently magnetized objects.

Walls of the pool are not magnetically neutral. Concrete walls typically contain steel reinforcement bars and metal mesh. As a result, they distort the geomagnetic field in their vicinity. Magnetic field lines are drawn into regions of higher permeability, causing a redistribution of the field that extends into the surrounding air.

In free space, the magnitude of the geomagnetic field changes very little. Near walls, however, the magnetic field can vary significantly, producing detectable gradients in both magnitude and direction.

Although magnetometers are often used to estimate orientation by assuming a constant field magnitude, this assumption breaks down near structures. The amplitude of the magnetic field changes because the wall acts as a passive magnetic modifier.

The same distortions that complicate magnetic navigation can be exploited to detect walls without vision. The key insight is that walls introduce magnetic field structures that do not exist in open space. By monitoring how the magnetic field changes as a function of position, it is possible to infer the presence and proximity of a wall.

3.7 Estimation of the North

If now, we assume that the magnetometer is perfectly calibrated and returns the corrected magnetic field expressed in the frame of the robot.

Assume that the calibrated magnetometer is far from the wall or other man-made structure. It returns the local magnetic field vector, *i.e.*,

$$\mathbf{y} = \mathbf{B}_{\text{Earth}} + \underbrace{\mathbf{B}_{\text{induced}} + \mathbf{B}_{\text{remanent}}}_{=\mathbf{0}}$$

in the robot's body frame. We now have to find where is the North.

Recall that

- The magnetic field does not lie entirely in the horizontal plane.
- The vector points toward *magnetic North*, not geographic North.

The notion of North direction is defined in the horizontal plane. The horizontal magnetic field vector $\mathbf{y}$ points toward magnetic North. If the robot operates on a flat surface, the z component can be neglected.

$$\psi = -\text{atan2}(y_2, y_1)$$

Exercises

EXERCISE 8.– *Flat calibration of the magnetometers*

For the DD-Boat, the motion is constrained to a nearly flat plane, making it sufficient to use only the horizontal components of the magnetic field. In this exercise, we present a complete 2D magnetometer calibration method, including both offline and online (while-driving) calibration, and explain the physical meaning of hard-iron and soft-iron perturbations. We assume:

- Vehicle roll and pitch are negligible
- Magnetometer axes (m_x, m_y) lie in the horizontal plane
- Earth magnetic field horizontal component has constant magnitude
- Calibration is performed without external heading reference

We first recall that in the absence of disturbances, the horizontal magnetic field measured by the sensor is

$$\begin{pmatrix} m_x \\ m_y \end{pmatrix} = \begin{pmatrix} h_0 \cos\psi \\ h_0 \sin\psi \end{pmatrix}$$

where h_0 is the magnitude of the horizontal magnetic field and ψ is the vehicle heading. Plotting (m_x, m_y) over all headings yields a circle centered at the origin.

1) What becomes this formula in case of hard-iron perturbations.

2) What do wet get in case of soft-iron perturbations.

3) What do wet get in case of both hard and soft iron perturbations.

4) To calibrate the magnetometer, we propose to drive the DD-Boat which encloses the magnetometers in circles to cover all headings. Give a method to calibrate the magnetometer.

5) Give the resulting formula to get the heading of the DD-Boat.

EXERCISE 9.– *3D calibration of the accelerometers*

A three–dimensional low–cost accelerometer embedded in the DD-Boat has to be calibrated, because such sensors inherently exhibit systematic errors. The most important of these errors is bias, which corresponds to a constant offset added to the true acceleration. This bias originates from manufacturing tolerances and exists in all low–cost MEMS accelerometers. In addition to bias, low–cost accelerometers suffer from scale factor errors, meaning that the sensor output does not perfectly match the true acceleration magnitude. Calibration exploits the fact that, when the robot is stationary, the only acceleration acting on the sensor is gravity. By placing the accelerometer in known orientations relative to the gravitational field, it is possible to estimate both bias and scale factors. Once these parameters are identified, the raw sensor measurements can be corrected so that the measured acceleration more closely matches the physical reality.

We assume that

$$\mathbf{x} = \mathbf{A}\mathbf{y} + \bar{\mathbf{x}}$$

where $\mathbf{x} \in \mathbb{R}^3$ is the acceleration vector which is measured, and $\mathbf{y}$ is gravity vector (oriented upward). We assume the boat static so that the acceleration corresponds to the gravity. The calibration step searches for an estimation of $\mathbf{A}$ and $\bar{\mathbf{x}}$.

We take four measurements $\mathbf{x}_x$, $\mathbf{x}_y$, $\mathbf{x}_z$, $\mathbf{x}_{-z}$, , with the robot, as illustrated by Figure 3.12. We do not care where the North is.

- Direction x : The robot is set vertically with its x axis looking upward
- Direction y : The robot is set with roll its y axis upward
- Direction z : The robot is laid flat with its z axis upward
- Direction $-z$: The robot is laid flat upside down with its z axis downward

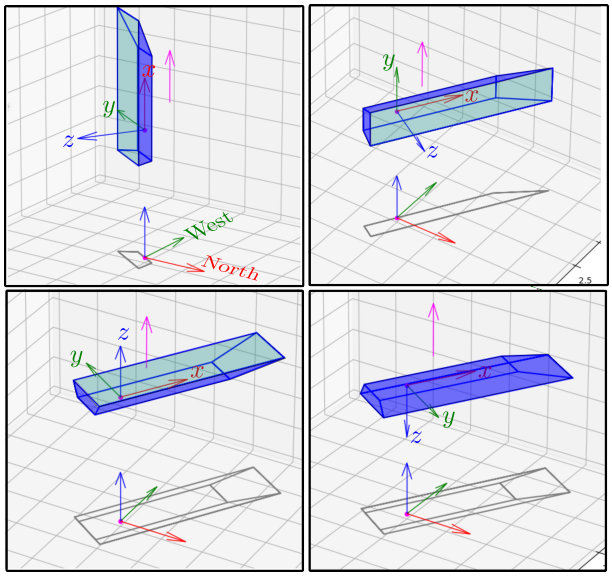


Figure 3.12: Procedure to calibrate the accelerometers

Deduce an algorithm to estimate $\mathbf{A}$ and $\bar{\mathbf{x}}$.

EXERCISE 10.– *Ellipsoid to sphere*

Transforming an ellipsoid to a sphere is an important step to be solved for the autocalibration of the magnetometers. Consider the linear transformation $\mathbf{x} = \mathbf{A}\mathbf{y} + \bar{\mathbf{x}}$ where

$$\mathbf{A} = \begin{pmatrix} 4 & 1 & 2 \\ 1 & 3 & 1 \\ 2 & 1 & 2 \end{pmatrix}, \bar{\mathbf{x}} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

where $\mathbf{A}$ has been chosen symmetric positive definite. Assume that $\mathbf{y}$ belongs to the unit sphere.

1) Show that $\mathbf{x}$ belongs to an ellipsoid and give the corresponding equation.

2) With Python, generate many points $\mathbf{y}(i), i \geq 0$ in a unit sphere and the corresponding $\mathbf{x}(i), i \geq 0$. Save all of them in a csv file.

3) From the $\mathbf{x}(i)$, provide a python code which estimates $\mathbf{A}$ and $\bar{\mathbf{x}}$.

4) Show how you can get an estimate $\hat{\mathbf{y}}(i)$ the $\mathbf{y}(i)$'s from the $\mathbf{x}(i)$'s.

5) Check by drawing the corresponding clouds of points. You should get a figure similar to the Figure 3.13.

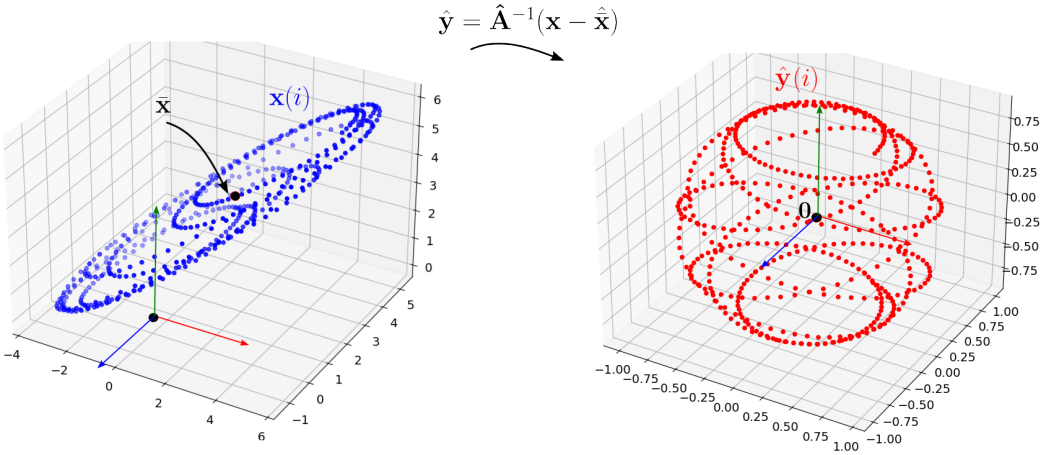


Figure 3.13: From the cloud of blue points, we get the unit sphere (red)

Chapter 4

Practice with the DD-Boat

This chapter is partly taken from the lesson of B. Zerr [14].

4.1 Introduction

The DD-Boat is a small autonomous surface vehicle designed as an experimental platform for robotics education. The DD-Boat is a twin-screw vessel without a rudder with a V-hull, steering is achieved through differential thrust. The standard configuration is outward-rotating, where the right-hand pitch propeller is mounted on the starboard (right) side and the left-hand on the port side. When you program the DD-Boat, please avoid going backward: water may enter inside the hull.

The robot integrates sensing, computation, communication, and actuation in a real outdoor environment, making it particularly well suited for introducing students to embedded robotics and mobile navigation. You need to learn how to

- interact with an embedded robotic system through a network interface,
- deploy and execute Python programs on a remote onboard computer,
- acquire and log GPS data,
- understand practical issues related to communication latency, blocking I/O, and network failures.

The DD-Boat architecture follows a classical mobile robotics design:

- an onboard computer (Raspberry Pi) running Linux,
- sensors (GPS, IMU),
- motor controllers for propulsion and steering,

- a wireless communication interface (WiFi).

All user programs are executed directly on the onboard computer. The student laptop acts only as a development and supervision terminal, not as a real-time controller. This separation reflects standard practice in field robotics.

4.2 Hardware

Inside the DD-Boat, the hardware is described by Figure 4.1.

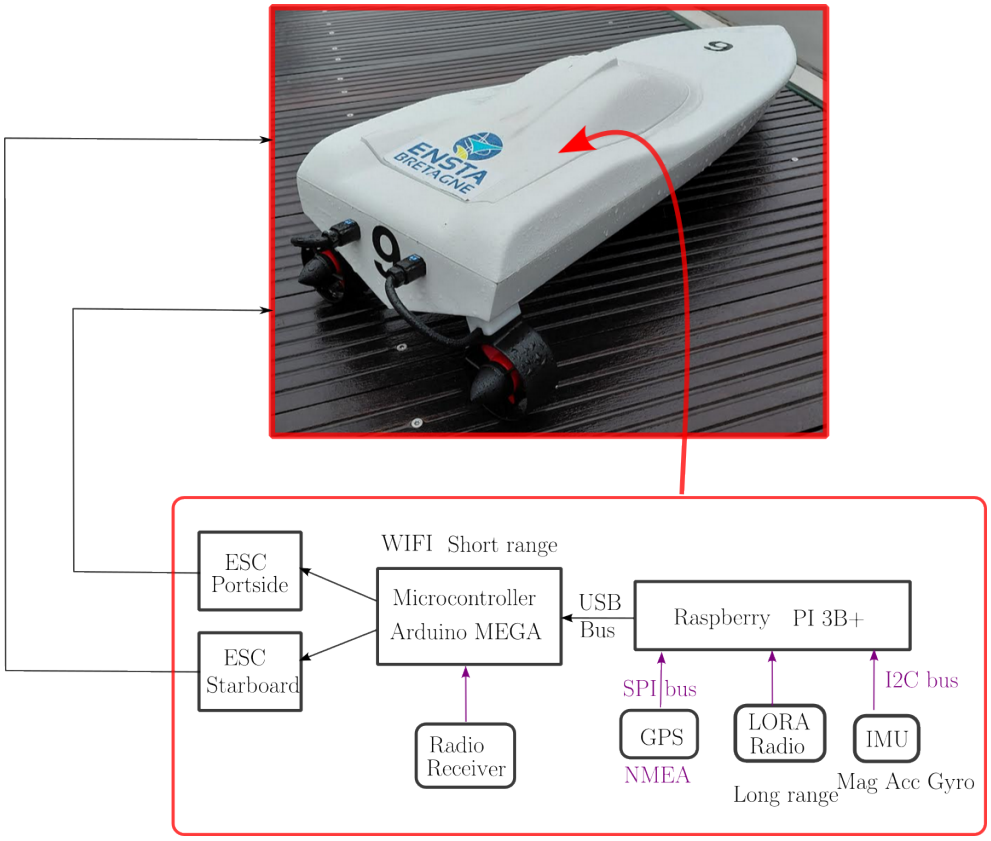


Figure 4.1: Hardware of the DD-Boat

The DD-Boat is equipped with two electric propulsion units, each driven by an *Electronic Speed Controller* (ESC), allowing differential thrust for maneuvering. An Arduino MEGA microcontroller is used for low-level real-time control of the motors and sensors. A Raspberry Pi 3B+ serves as the high-level processing unit, handling navigation, communication, and data management. Positioning is provided by a GPS module outputting NMEA sentences for accurate localization. The orientation is obtained by the IMU (Inertial Measurement Unit). Long-range communication is achieved using a LoRa radio link, while Wi-Fi is available for short-range configuration and monitoring. The system architecture separates control and computation to improve reliability and flexibility.

Batteries

The DD-Boat is powered by two 2S LiPo (Lithium-Polymer) batteries, one supplying the motors and the other supplying the onboard electronics. Before starting a mission, the voltage of both batteries must be checked. For a short (standard) mission, the battery voltage should be greater than 7.7 V, while for a long mission it should be greater than 8.0 V. The batteries must be securely attached using Velcro straps on both sides of the electronic box. Once mounted, plug the batteries into the electronic box using the appropriate connectors. Power on the system using the ON/OFF switch located on top of the electronic box. When powered, the yellow LED on the electronic box should illuminate, indicating that the electronics are active. A short melody will then be played, indicating that the motors are powered. After the melody has played three times, the DD-Boat is ready to be controlled.

In a Lithium-ion (Li-ion) or LiPo, battery pack (as in the DD-Boat), a multiple individual cells are wired together. While the main thick wires provide the power, the smaller balance connector (see Figure 4.2) ensures each individual cell stays healthy.

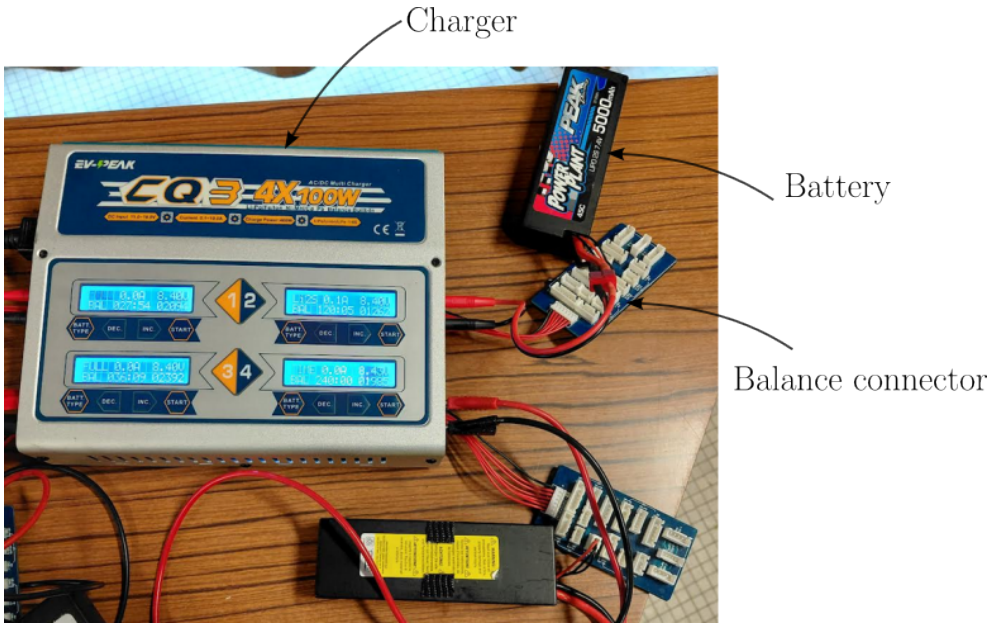


Figure 4.2: The battery on the top right can be charged to a capacity of $5000mAh$

When charging a 2-cell lithium-polymer (2S LiPo) battery, it is important to understand both the electrical characteristics of the battery and the role of the charger. A 2S LiPo battery consists of two lithium-polymer cells connected in series. Each cell has a nominal voltage of about 3.7 V, so the nominal voltage of the pack is 7.4 V. During charging, each cell must reach a maximum voltage of 4.2 V, which means the fully charged pack reaches 8.4 V. Charging beyond this voltage can damage the battery and presents a safety risk.

The CQ3-type LiPo charger is designed as balance chargers. When a 2S battery is connected to the charger, the balance connector is inserted into the 2S charging port. The charger detects the

two-cell configuration through the number of wires and the voltage between them. During charging, the charger applies the standard used for lithium batteries. In the first phase, the charger delivers a controlled current to the battery while the voltage gradually increases. This stage is called the constant-current phase. Once the battery approaches its maximum voltage of 8.4 V, the charger transitions to the constant-voltage phase, maintaining the voltage at this limit while the charging current progressively decreases.

A key function of the charger is cell balancing. In a multi-cell battery, the individual cells may not have exactly the same voltage due to manufacturing tolerances, aging, or uneven discharge. If one cell reaches 4.2 V earlier than the others, the charger must reduce the current or dissipate energy through balancing circuits to prevent overcharging that cell. The balance connector provides the measurements needed for this process. Proper balancing ensures that all cells end the charging process at approximately the same voltage, which improves battery lifespan and safety.

When the charging current falls below a small threshold during the constant-voltage phase, the charger determines that the battery is fully charged and stops the process. At this point, the pack voltage is approximately 8.4 V, corresponding to 4.2 V per cell. Charging beyond this limit is avoided because lithium-polymer chemistry is sensitive to overvoltage, which can lead to degradation, swelling, or thermal runaway.

Remote control

Remote controls (see Figure 4.3) are assigned according to the color tape attached to the electronic box (black, grey, or green). Make sure to select the remote control that matches the color of your DD-Boat. Once the correct remote is selected, choose your DD-Boat using the switches on the transmitter. The left joystick controls the propulsion power and should initially be set to low or medium. The right joystick controls the direction of the boat, allowing forward, backward, left, and right movements. Switch at the top has two positions: the up position enables remote-control mode, while the down position enables autonomous mode. Switches A, B, and D each have three positions and together define the identification code of the DD-Boat in base 3. This code corresponds to the boat number minus one, expressed in base 3. For example, DD-Boat 12 has a code of 11 in decimal, which is 102 in base 3 ($1 \times 3^2 + 0 \times 3^1 + 2 \times 3^0 = 11$). The corresponding switch positions are $A = 2$, $B = 0$, and $D = 1$.

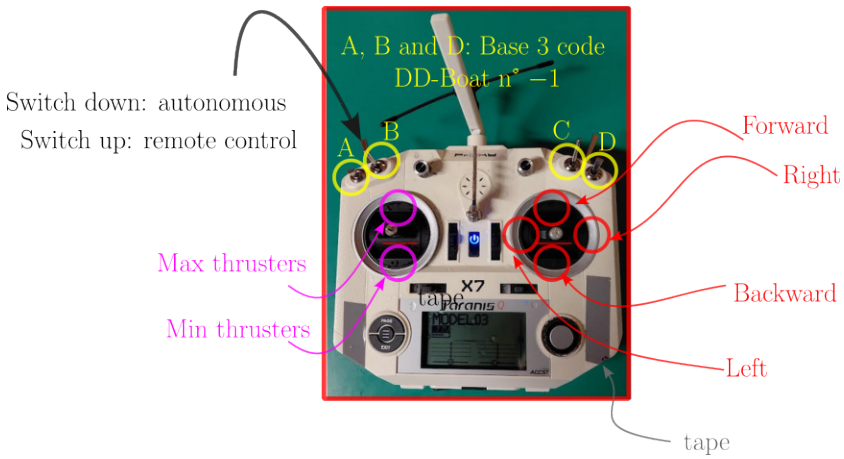


Figure 4.3: Remote control of the DD-Boat; here the color tape is grey

4.3 Talking to the DD-Boat

4.3.1 Network: DARTAP

Modern local area networks, particularly those used in educational robotics, often require a hybrid approach to balance legacy compatibility with high-speed data transmission. For the experiments, we built a dual-router with both 2.4 GHz and 5 GHz frequencies, alongside the security implications of WEP and WPA2 protocols. The configuration of two routers connected via an Ethernet cable creates a single, unified logical network from two distinct physical layers. In this setup, the primary (older) router typically handles the 2.4 GHz band, which is characterized by longer wavelengths that penetrate obstacles effectively but offer lower data rates. The secondary (newer) router handles the 5 GHz band. and delivers significantly higher throughput and reduced latency. Because these devices are linked by a physical cable, they share the same broadcast domain. This means that your computer connected to the 5 GHz high-speed signal can seamlessly address a robot connected to the 2.4 GHz signal. The routers act as bridges, translating wireless frames into wired Ethernet frames and back again, allowing devices on different frequencies to communicate as if they were on the same wire.

A critical aspect of managing such a network is the implementation of security protocols, which govern how data is encrypted as it travels through the air. The two most common standards encountered in legacy and modern systems are WEP and WPA2.

- WEP (Wired Equivalent Privacy) was the original security standard for Wi-Fi. WEP was retained for the DD-Boats because older microcontrollers lack the processing power to handle modern encryption. However, WEP is fundamentally insecure.
- WPA2 (Wi-Fi Protected Access 2) replaced the aging WEP standard. It provides a robust defense against unauthorized access.

Our network is named DARTAP. It provides a simple IP-based communication channel between the laptop and the onboard computer.

- The WiFi password for DARTAP is `ABACA`.
- Your computer should be connected to DARTAP-5GHZ, password ABACA-5GHZ
- Each DD-Boat has a static IP address: `172.20.25.2xx`, where xx is the two-digit boat number.

Using static IP addresses simplifies network configuration and avoids the need for service discovery mechanisms, which are often unreliable in outdoor environments.

Before executing any program, network connectivity must be verified. In your computer, open a terminal T1 and write

```
ping 172.20.25.2xx
```

A successful ping confirms:

- the WiFi link is established,
- IP routing is correct,
- the onboard computer is powered and responsive.

4.3.2 SSH

SSH is used to remotely access and test the hardware components of a DD-Boat. We need to understand both the general principles of remote access and the practical commands used to verify that the boat's sensors and actuators are functioning correctly. SSH, which stands for *Secure Shell*, is a network protocol that allows a user to securely connect to another computer over a network. Once connected, the user can control the remote computer through a command-line interface as if they were physically sitting in front of it.

SSH is widely used because it encrypts all communications between the two machines. This means that passwords, commands, and data exchanged during the session cannot be easily intercepted or read by third parties. For this reason, SSH is the standard tool for remote administration of servers, embedded systems, and robots. SSH allows us to connect from our personal computer to the onboard computer (here a Raspberry Pi inside the DD-Boat) and execute programs that interact with the boat's hardware.

To connect to a DD-Boat, open a second terminal named T2 from our own computer. Run the following command:

```
ssh ue32@172.20.25.2xx
```

The username `ue32` (*Unité d'Enseignement 3.2*) refers to the user account on the Raspberry Pi inside the DD-Boat. The password is `ue32`. Once authenticated, all subsequent commands are executed directly on the DD-Boat.

In your computer, you should thus have two terminals open. T1 is your own computer and T2 acts as a remote terminal for the boat.

4.4 First program

Recall that you should have now two terminals T1 and T2 open . T1 corresponds to your computer and T2 to the Raspberry of your DD-Boat.

After connecting with `ssh`, you are in T2 with the Linux of the DD-Boat. Write:

```
$ ls -l
```

to see the files. Each team creates its own directory:

```
$ mkdir DroneDupond
$ cd DroneDupond
```

where Dupond is the family name of the Team's leader. This directory-based organization avoids conflicts between teams and mirrors common multi-user development practices. The `cd` (change directory) command is used to move within the file system.

The first Python program serves as a *deployment and execution test*. It is saved in `test.py` and contains

```
print("Hi!")
```

This trivial script verifies:

- Python is correctly installed on the DD-Boat,
- file transfer and execution permissions are correct,
- the `ssh` terminal correctly displays program output.

The file is transferred from Terminal T1. To copy from your computer to the DD-Boat write:

```
$ scp test.py ue32@172.20.25.2xx:DroneDupond/
```

The program is now on the DD-Boat. Go to Terminal T2 and run the program with the command : `python3 test.py`. You should see the message "Hi!" on the screen of T2.

4.5 Drivers

A driver is a low-level software component that acts as a translator between the Operating System (OS) or a microcontroller and a specific piece of hardware. A driver manages physical signals (Voltage, PWM, Clock cycles) and is often located the Kernel space (OS) or as Firmware (on the Arduino). A driver is often used with an API (Application Programming Interface) which is a set of defined methods that allow one software to communicate with another. The API simplifies the use of the driver. It provides high-level commands such as `motor.stop()` or `gps.get_coords()`. In robotics, we often confuse a driver with the associated API. This confusion will be done in this lesson.

In the DD-Boat, each driver is implemented as an independent Python script. This modular approach makes the system easier to understand, test, and maintain. Several drivers are provided with the DD-Boat software. The inertial measurement unit (IMU) driver handles the magnetometer, accelerometer, and gyroscope. A second driver is dedicated to the GPS receiver and processes standard NMEA messages. Motor commands are sent through an Arduino board using a serial communication channel. Finally, a simple radio driver is available for monitoring purposes only, using a LoRa radio link. All drivers follow the same general philosophy: read data from a sensor or send commands through a serial interface, then expose this functionality to higher-level programs.

Installing and Testing the Drivers. To begin working with the DD-Boat drivers, you can download these drivers here

```
https://webperso.ensta.fr/jaulin/drivers-ddboat-v2.zip
```

The files in `drivers-ddboat-v2` contains the software programs that allow the Raspberry Pi to communicate with the boat's sensors and actuators, such as the GPS, IMU, and motors.

You can inspect the Python files and run test scripts to verify that each driver behaves correctly on your boat.

GPS. To test the GPS module, run from Terminal T2 the Python program `gps_driver_v2`:

```
$ python3 gps_driver_v2.py
```

This program communicates with the GPS sensor and displays the position data, such as latitude and longitude. If the GPS is working correctly, the measurements should appear consistently after a short initialization time. Once the first measurements appear correct, the program can be stopped by pressing `Ctrl+C`. This keyboard shortcut sends an interrupt signal that safely terminates the running program.

Note that the GPS measurements are:

- noisy (meter-level accuracy),

- low-frequency3 (typically 1–10 Hz),
- subject to dropouts in obstructed environments, like the ENSTA robotics pool.

Some additional Python packages are required to handle GPS data and coordinate transformations.

The GPS receiver provides NMEA messages, in particular the `GPGLL` message, which contains latitude and longitude information. These coordinates are stored in a GPX file so that they can be visualized using mapping software. In NMEA format, coordinates are expressed as degrees and decimal minutes. To convert them into decimal degrees, the following formula is used:

$$\text{degrees} = DD + \frac{mm.mmmm}{60}$$

Once expressed in decimal degrees, coordinates can be projected into local coordinates.

IMU. The IMU (Inertial Measurement Unit) is a 9 degrees of freedom sensor (accelerometer, gyroscope, magnetometer). It gives

- the acceleration in 3 directions (x, y, z) on 16 bits,
- the angular speed in 3 directions (x, y, z) on 16 bits,
- the magnetic field in 3 directions (x, y, z) on 16 bits.

The IMU is a low cost IMU-9v5 from Pololu https://www.pololu.com/product/2738 (cost:30 euros) which is a single, tiny breakout board (roughly 0.8" $\times$ 0.5") that comes with both of two specific chips:

- LSM6DS33: a 3-axis gyroscope and 3-axis accelerometer
- LIS3MDL: handles the 3-axis magnetometer

The IMU is tested using the following command (in Terminal T2):

```
$ python3 imu9_driver_v2.py
```

When this program is running, the displayed values should change if the boat is moved or rotated. This confirms that the motion sensors are operating properly.

How to get the IMU data :

```
import sys, os, time
sys.path.append(os.path.join(os.path.dirname(__file__), 'drivers-ddboat-v2'))
import imu9_driver_v2 as imudrv
imu = imudrv.Imu9IO() # create an Arduino object
xmag,ymag,zmag = imu.read_mag_raw()  # magnetic raw data (raw = not calibrated !)
xaccel,yaccel,zaccel = imu.read_accel_raw() # accelerometer
xgyro,ygyro,zgyro = imu.read_gyro_raw() # gyroscope
```

Motor. Propellers are powered by 2 brushless motors controlled by an electronic speed controller (ESC). The ESC are connected to an Arduino MEGA 2560 micro controller board. After DDBoat startup, wait for the melody to play 3 times before commands sent to the ESC become effective.

To test the motors, the following command is used:

```
$ python3 arduino_driver_v2.py speed_left speed_right
```

In this command, `speed_left` and `speed_right` represent the speeds of the left and right motors, respectively. These values must be integers in the range from -255 to $+255$. Positive values correspond to forward motion, negative values to backward motion, and zero stops the motor.

By choosing different values for the two motors, it is possible to test forward motion, reverse motion, and turning behavior. As with the other tests, the motors can be stopped safely by pressing `Ctrl+C`.

Example of in place rotation for 2 seconds :

```
import sys, os, time
sys.path.append(os.path.join(os.path.dirname(__file__), 'drivers-ddboat-v2'))
import arduino_driver_v2 as arddrv
ard = arddrv.ArduinoIO() # create an Arduino object
left_speed = 100
right_speed = -left_speed
ard.send_arduino_cmd_motor(left_speed,right_speed)
time.sleep(2)
ard.send_arduino_cmd_motor(0,0) # stops
```

4.6 Screen

A critical aspect of field robotics is dealing with unreliable communication.

- WiFi coverage is limited and intermittent.
- SSH terminals rely on continuous TCP connections.
- Blocking output operations (e.g. `print`) can freeze a program if the connection is lost.

This behavior is particularly dangerous for robots, as it can stop control loops. To avoid this, programs must be executed inside a detached terminal using `screen`. This ensures that the program continues running even if WiFi is lost. Control and data acquisition remain autonomous. A screen session is started with a chosen name, for example:

```
$ screen -S sesddboat
```

where `sesddboat` is the name (or "Session ID") given to a specific screen session. This opens a new terminal managed by `screen`. You can verify that the session is active by listing existing sessions:

```
$ screen -ls
```

Run the Python program on the DD-Boat. Detached the session from the terminal T2 using the key sequence `Ctrl+A` followed by `D`. The program continues to run in the background. Later, when the WiFi connection is restored, the session can be resumed with:

```
$ screen -r sesddboat
```

where 'r' means *resume*. When the work is finished, the screen session can be completely stopped with:

```
$ screen -X -S sesddboat quit
```

Example: Logging GPS Data

As an example, a GPS logging program can be started inside a screen session:

```
$ screen -S sesgps
$ python3 tst_gpx.py
```

After detaching the session, the boat can move outside the WiFi coverage area while recording GPS data. Once finished, the session is resumed, the program is stopped, and the generated file can be renamed for later use.

4.7 GPS

4.7.1 Time to first fix

Often excessive delays in obtaining a GPS position fix (initialization). The delay in obtaining a position, known as the *Time-To-First-Fix*, is influenced by two main factors: electromagnetic interferences and ephemerides.

- Electromagnetic Interference (EMI): The Raspberry Pi is a high-speed digital device. The CPU, RAM, and specifically the HDMI clock produce significant radio frequency noise in the 1.5 GHz range, which overlaps with the GPS band. When possible, increase the physical distance between the antenna and the Raspberry (minimum 10–15 cm) or use a ground plane (copper/aluminum foil) underneath the antenna to shield it from unwanted radiation.

- To compute a position quickly, a GPS receiver needs a precise current time, a last known position and ephemerides (satellite orbital parameters). Without this knowledge, the module performs a cold start every time it is power-cycled. This means that the sensor must download the ephemerides from the satellites which takes approximately 12 minutes under ideal conditions.

A program named `testgpx.py` is made available in `/drivers-ddboat-v2/utilities`. This function allows to record GPS data with your DD-Boat and save it in file.

4.7.2 Accuracy

A standard GPS has an accuracy limited to about 3 to 5 meters which can be consider as large for the DD-Regatta. To be more precise, we may use RTK (Real-Time Kinematic). RTK is a technique used to enhance the precision of position data derived from satellite-based positioning systems, bringing that margin of error down to 3 centimeters. Whereas standard GPS calculates the position by measuring how long it takes a signal to travel from a satellite to your receiver. RTK uses a second, stationary source, the *base station*, to correct errors in real-time. The base station, fixed at a known location, can see the difference between its true location and what the satellites are telling. It calculates the error caused by the atmospheric interference (the ionosphere and troposphere) that slow down satellite signals. The base station broadcasts its correction data to the DD-Boat via an internet connection. The GPS then takes the satellite data it is receiving and subtracts the errors identified by the base station.

4.8 Communication

Modern robotic systems rarely operate in isolation. Even relatively simple robots often need to communicate with external systems for supervision, coordination, or data analysis. This communication is typically implemented using networking concepts, among which the client–server model is one of the most fundamental. In this section, we introduce the client–server paradigm from a robotics perspective, focusing on an architecture where DD-Boats act as clients and connect to a central system. This approach is particularly useful when dealing with multiple robots interacting with a shared interface or control system.

Robots interact with the world through sensors and actuators, but they also need to interact with other digital systems. These interactions include sending measurements, receiving commands, reporting status, and coordinating with other robots.

Client–Server Model: A server is a program that waits for incoming connections and provides services. A client is a program that initiates a connection and requests those services. A central system, for example, an HMI (see the following section) or a virtual environment, acts as a server. DD-Boats may connect to one or several servers and exchanges information with it.

Sockets. A socket is a software abstraction that enables communication between two programs over a network. To establish a connection, two elements are required: an IP address, which identifies a machine on the network, and a port number, which identifies a specific application running on that machine. When two programs communicate, each of them uses a socket. Robotic applications rely on TCP sockets which provide reliable, ordered communication, ensuring that messages are delivered correctly and in sequence. They may also use UDP that are fast but unreliable with no guarantee of delivery. Note that you should create one socket per connection and reuse it, not one per message.

Communication flow. The sequence of operations is as follows. The central server starts first and waits for incoming connections. Each robot then starts independently. It creates its own socket and initiates a connection to the server. Once connected, the robot can periodically send data such as sensor readings or status information. The server can respond with commands or configuration updates. This structure naturally supports multiple robots, since the server can accept several connections at the same time.

Programming with Python. Python provides a simple interface for socket programming through its standard `socket` library. On the server side, the program creates a socket, binds it to a port, and listens for incoming connections. When a robot connects, the server accepts the connection and communicates with it. On the robot side, the program creates a socket and connects to the server using its IP address and port. Once the connection is established, the robot can send messages and receive responses. Messages are transmitted as sequences of bytes. In practice, these bytes are often encoded strings (for example, using UTF-8) or structured formats such as JSON.

Virtual regatta. A virtual regatta is proposed to test your controller. In our network (DARTAP-5GHZ), the server is assigned the IP address 172.20.24.112, which belongs to a private IP range commonly used in internal networks. The server shows the robotics pool with somme DD-Boats using unity. To participate to the virtual regatta, you should connect to this server from your own machines, whether you are using Windows or Linux systems. An IP address is made up of two parts: the network portion and the host portion. The subnet mask determines how these two parts are divided. In this case, the subnet mask is written as 255.255.255.0 (also known as /24). This means that the first three octets (172.20.24) identify the network, while the last octet identifies individual devices (hosts) on that network. As a result, all devices that want to communicate directly with the server must have an IP address in the same range, for example: 172.20.24.1 or 172.20.24.50 or 172.20.24.200. All these devices share the same subnet and can communicate without needing a router. The subnet mask 255.255.255.0 allows for up to 254 usable host addresses (from 172.20.24.1 to 172.20.24.254), excluding the network address (172.20.24.0) and the broadcast address (172.20.24.255). In summary, as long as you computer is configured with an IP address in the same subnet (172.20.24.x) and the same subnet mask (255.255.255.0), they will be able to access the server directly over the network.

4.9 Human-Machine Interface (HMI)

In the world of robotics, Human-Machine Interface (HMI) allows us to communicate instructions to a robot and allows the robot to provide feedback to us. Without an HMI, a robot is just a black box executing pre-programmed code with no way for a user to intervene or monitor its progress. HMI translates complex machine data (like estimated position, batteries, heading, and alerts) to the user screen. The HMI also sends commands to the robot's for teleoperation. As illustrated by Figure 4.4, you will need an HMI running in your computer as a server to visualize the collected data of your DD-Boat. A simple way to communicate between the robot and your computer is to use Python sockets to exchange messages. For instance, as a client, the DD-Boat may send you its position, its heading so that you can draw the DD-Boat in the virtual pool.

To build an HMI, a client-server architecture has to be used. The DD-Boat acts as a client and sends data and you computer acts as a server. All data is sent as a simple line of text. For instance, the DD-Boat sends values separated by spaces, *e.g*, 1.2 0.5, 90 to represent the position and the heading.

Figure 4.4: An HMI: from your computer, you can see the state of your DD-Boat and control it

Exercises

Exercise 11.– *Collect GPS data*

Guide your DD_Boat along a route of several hundred meters in the campus while recording its GPS estimated position over time. The route should include both open areas and regions near buildings or trees so that changes in localization quality can be observed.

Export the recorded data to a GPX file and visualize in Google Earth.

Overlay the trajectory on satellite imagery and annotate the start and end points, major turns, and recognizable landmarks. Visual inspection should be used to identify obvious drift, discontinuities, or systematic deviations from the expected path.

Repeat several times, either by walking the same path again or by reversing direction.

Differences between runs should be analyzed and related to environmental conditions and motion behavior.

Exercise 12.– *Navigation using magnetism only*

Program the DD-Boat navigating in the pool using only its onboard magnetometer and motor control. Verify experimentally that the magnitude of the measured magnetic field increases when the boat approaches the pool walls. This spatial variation of the magnetic field will be used as the sole navigation cue. Your objective is to design a control law that makes the boat converge to a stable closed trajectory bouncing the pool wall, without ever touching it. The trajectory must be repeatable over many cycles and robust to small disturbances. No position estimation, distance sensors, or external localization systems are allowed.

The boat should move forward continuously while adjusting its heading based on the measured magnetic field magnitude. Intuitively, when the measured field becomes too large, the boat should turn away from the wall, and when it becomes too small, it should turn toward it. You are free to choose the control gains and the reference magnetic field value that corresponds to a safe distance from the wall.

During the experiment, record the magnetic field magnitude as a function of time and observe whether it converges to a steady oscillation. You should verify experimentally that the resulting

motion corresponds to a stable limit cycle and discuss why such stability emerges from the feedback between the magnetic field gradient and the control law.

This exercise illustrates how environmental physical fields can be exploited for navigation without explicit mapping or localization, a principle commonly used in bio-inspired and minimal-sensing robotic systems.

EXERCISE 13.– *DD Regatta*

In the pool, there exists two gates: the green gate delimited by $\mathbf{a} : (6, 5)$ and the red gate delimited by $\mathbf{b} : (6, 15)$, as shown on Figure 4.5. Each DD-Boat has to start either from the South border or from the North border. Each time the robot crosses one gate in the right direction, the team gets one point. In the wrong direction, the team looses one point. The competition lasts for 5 minutes and you should get as many points as possible. The starting time is given by the jury. Several boats can compete together in the pool. A video should be done and a log file should be given to the jury.

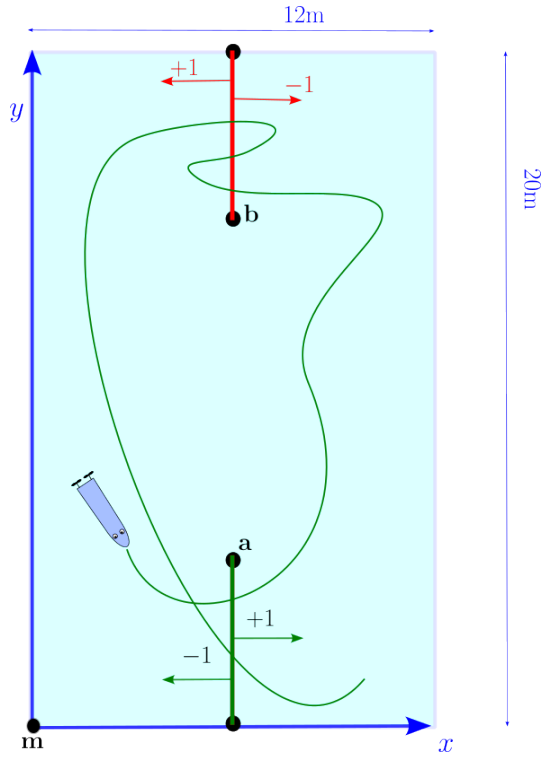


Figure 4.5: For the green trajectory, the DD-Boat got $1 + 1 - 1 + 1 + 1 = 3$ points

Bibliography

[1] T. Kailath. *Linear Systems*. Prentice Hall, Englewood Cliffs, 1980.

[2] L. Jaulin. *Automation for Robotics*. ISTE editions, 2015.

[3] H.K. Khalil. *Nonlinear Systems, Third Edition*. Prentice Hall, 2002.

[4] J.J. Slotine and W. Li. *Applied nonlinear control*. Prentice Hall, Englewood Cliffs (N.J.), 1991.

[5] I. Fantoni and R. Lozano. *Non-linear control for underactuated mechanical systems*. Springer-Verlag, 2001.

[6] L. Jaulin. *Représentation d'état pour la modélisation et la commande des systèmes (Coll. Automatique de base)*. Hermès, London, 2005.

[7] L. Jaulin. *AutoMOOC, Linear Control with a State Space approach* , `https://webperso.ensta.fr/jaulin/automooc.html`. ENSTA-Bretagne, 2019.

[8] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497, jul 1957.

[9] F. Boyer, M. Alamir, D. Chablat, W. Khalil, A. Leroyer, and P. Lemoine. Robot anguille sous-marin en 3d. In *Techniques de l'Ingénieur*, 2006.

[10] C. Chevallereau, G. Bessonnet, G. Abba, and Y. Aoustin. *Les robots marcheurs bipèdes ; Modélisation, conception, synthèse de la marche, commande*. Hermès-Lavoisier, Paris, 2007.

[11] T. Murata. Petri nets : properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[12] T. Fossen. *Guidance and Control of Ocean Vehicles*. Wiley, New York, NY, 1995.

[13] Stephen Blundell. *Magnetism in Condensed Matter*. Oxford University Press, Oxford, 2001.

[14] B. Zerr. Getting started with dd-boat. Lesson in Robotics, 2025.