Control of a catamaran

Ulrich CAMILLE, Mamadou DEMBELE

Summary

Chapter 1

Introduction

Chapter 2

Embedded Software

The purpose of this project is to create a software to control the catamaran we have in class.The catamaran should be able to follow a curve drawn on a map and go precisely at his final destination (x,y) but also manage his angle. The GUI should be able to reference the state of the catamaran but also to be able to define its path on a map and make him follow the path and lead him to his final destination.

2.1 Boat configuration

engines configuration on the catamaran is important to define for the simulation on our software.Indeed, the catamaran must be really handy because he must be able to stay precisely on at his final destination (x,y) but also his angle that should be static too.

we proposed configuration : -Differentiated push, that is to say engine on the side and also a rotation on z axis for each engine . -Rear and front engine with also rotation on z axis for each engine.

this boat has only 3 degree of freedom (x,y, rotation z). Then our solutions are both overactuated, so our order for the boat will be more complicated to calculate however our system will be more handy for navigation.

2.2 Block diagram

this schema represent the file streams in our project, and the different programming language used.

The GUI and python codes are all on an "external" computer to the system, only ROS is embedded on the CPU (Boat) with files created on an external machine then transmitted by SSH on the embedded CPU, this saves the CPU in long and complex calculations.

2.3 Application

The developed application allows to define the route to be taken by the boat in a discrete way. For the moment the application can start only thanks to Qt Creator, but eventually it will have to be compiled and executable on any machine (without Qt Creator). We started from a basic structure of a geolocation service and then to this one we added a type of marker : marker, and also a new function that will allow to extract the list of waypoints in a file. The difficulty here has been to understand how to do this because the GUI is coded in Qml so the functions are linked to signals, a characteristic specific to Qml, so how to link this signal to a C++ code ? For

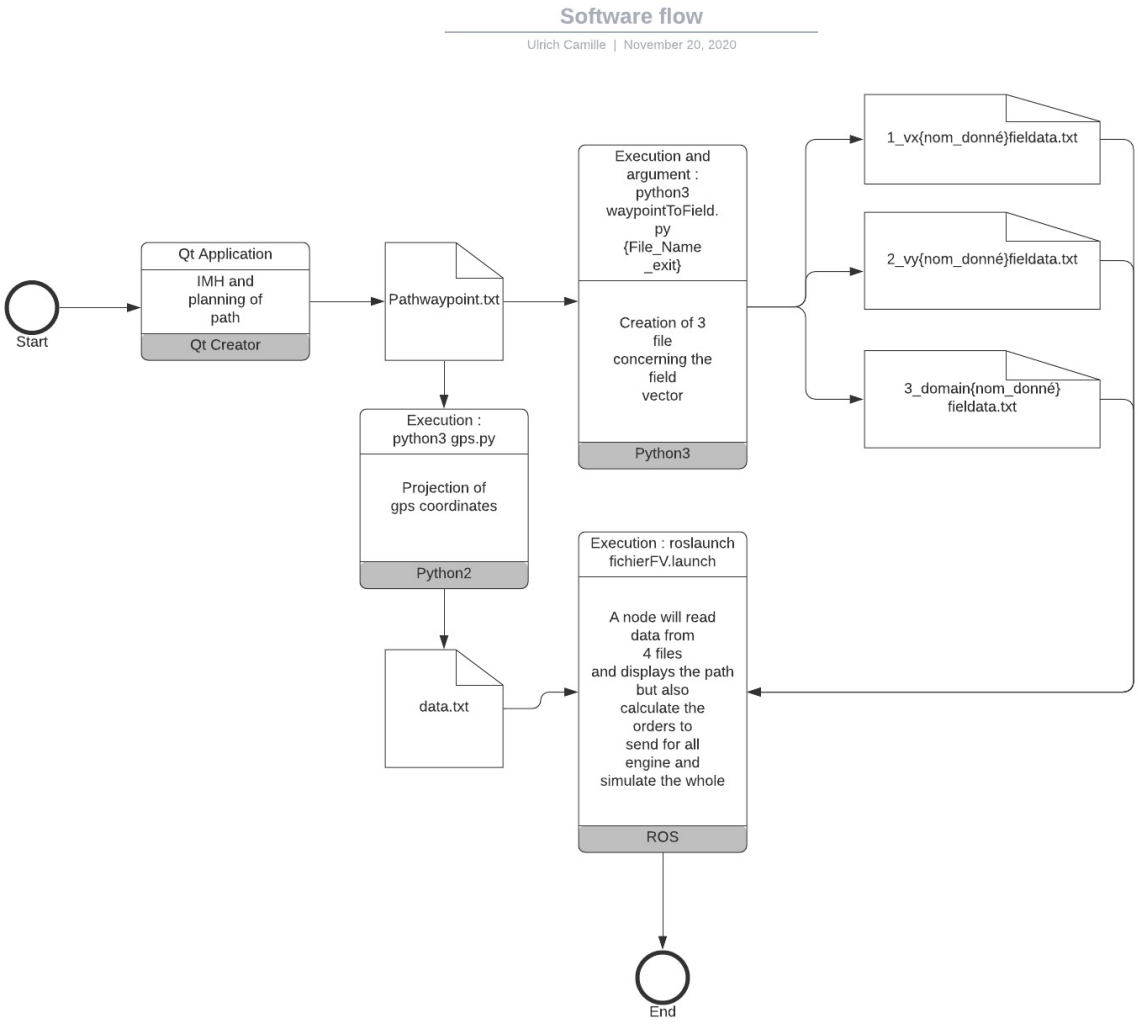

FIGURE 2.1 – Diagram flow software

that the Qml file can only include C++ objects/structure in its code, so the way to call a C++ function in a Qml code is to indirectly call this function by integrating it as a method of a C++ object that can be integrated in the Qml code, the object is the intermediary. Here the object is called FileIO, and so that it can be used in Qml files, we initialize it in the main file and then we use this method :

engine.rootContext()->setContextProperty("fileIO", &fileIO) ;

and it is the write() method of the FileIO object that will create a file from a string obtained by the GUI retrieved by the qml code.

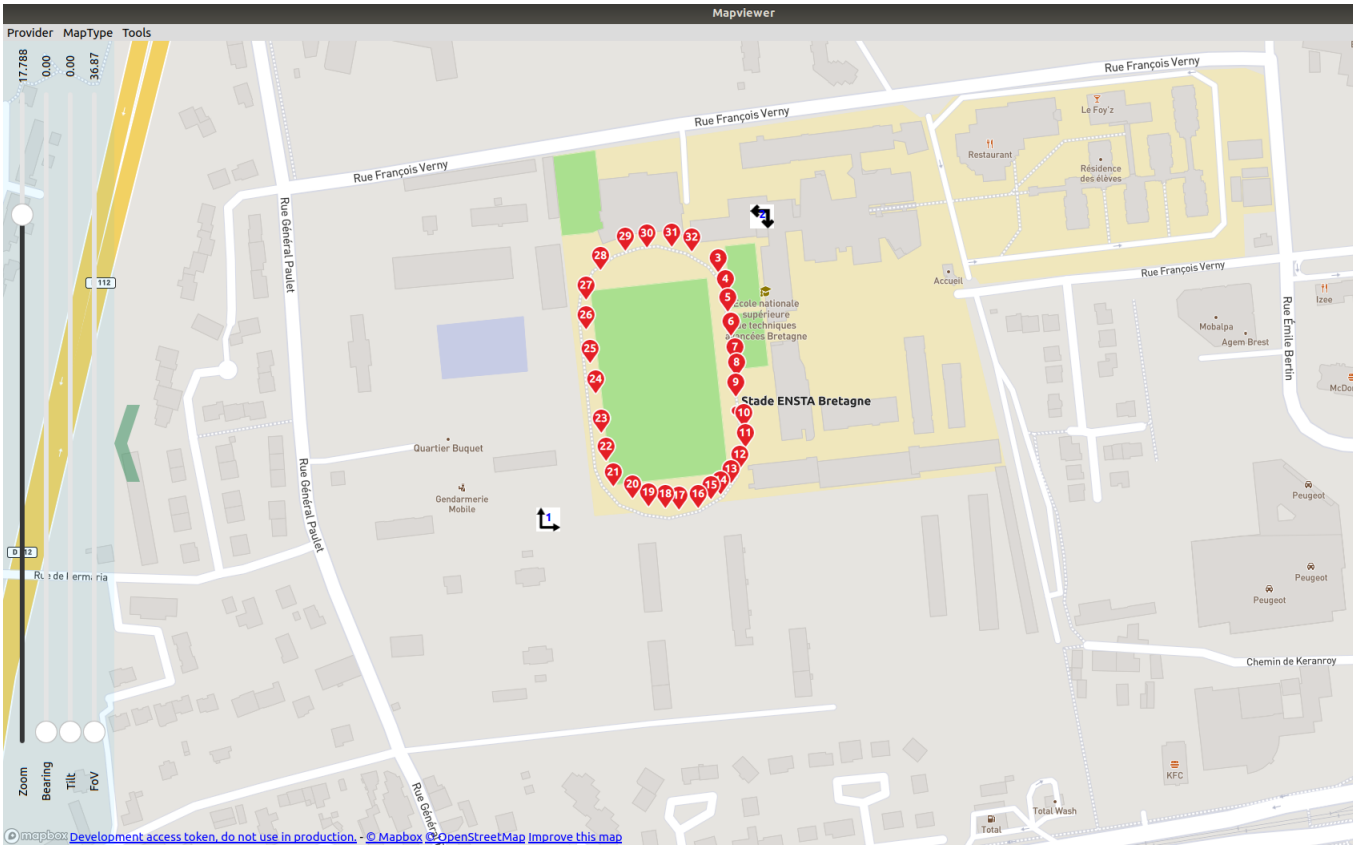

FIGURE 2.2 – Qt application screen

in the window menu :

- The "provider" menu just allows you to change the map and the geolocation system.
- In order to create a trajectory, you just have to add markers by staying pressed at the desired place, a small menu appears, then you have to press "add marker...".
  The first two markers are markers, between them they form a square which will be the definition of the domain of the velocity vector field. the other markers will constitute the waypoints to be followed
- In the "Tools" menu, a function named "Path creation" will create a file in the directory

The output file is a "Pathwaypoint_data_GPS.txt" file containing the list of waypoints but also the two reference points which are the first two points of the list and allows to set a reference point (the middle between the two points is the point (0,0) of the projection to a Cartesian landmark thanks to the python file gps.py (to be executed with the following command :

python2 gps.py [namefile]

which gives in output another file "[namefile]_data_cartesien.txt" which is created directly in the ROS package concerned by these data.
The output file is also argument to create the speed vector field which is executed with the following command :

python3 waypointToFieldvector.py [namefile]

This will create 3 files directly in the ROS package concerned. These data hold every speed instruction for each couple (x,y) in the discretized domain.

2.4 Field vector

The calculation of the vector field allows a robot that would follow it, to tend towards the desired path by giving speed instruction to the robot. To carry out this calculation for each point (x,y) we look for the nearest point of the curve and we approximate the derivative with a diagram (of order 1), then in order to make it converge the speed vector towards the curve, we carry out a rotation of the vectors between 0 and $\pm\ \pi\ /2$, the sign depends on the determinant between the vector derived from the point concerned and the distance vector between these two points. the angle depends on the distance between the two points concerned $\text{limits}_{x\to+\infty} distance = \pi/2$, so at infinity the velocity vector converges it with a velocity vector perpendicular to the derived vector. The complexity of this algorithm approaches the n^2 which is not reasonable, with n, the number of division of the domain and so affects the accuracy.

That's why the embedded CPU will never have to calculate all this information and for that the data is written on a file that only needs to be transmitted in SSH to the embedded CPU from a machine.

2.5 From speed vector to engine instruction

2.5.1 1st method

In a first step we simply realized a simple proportional derivative order with p1,p2 ranging from -200 to 200 N and theta motors ranging from $\pi\ /2/2$ to $-\pi\ /2$

In the configuration, with front and rear engine, the power still has a more special control because in order to avoid that the boat's speed also influences its ability to rotate, a factor in (1+ coef*sawtooth(difference_cap)) has been added while limiting it to its maximum value of about 200 N so when the engines are opposite, the engine thrust increases to make it converge faster towards its heading. (sawtooth is a function that takes 2 angles and then return the angle difference always between $[-\pi\ ;\ \pi\ ]$

2.5.2 2nd method

The speed vector instruction is not sufficient, in fact it translates a speed instruction by a command on the 4 actuators of the catamaran (2 motors, 2 servomotors). To do this, we started from the equations of states

$$\begin{cases} x_{dot} = cos(psi) * u - sin(psi) * v \\ y_{dot} = sin(psi) * u + cos(psi) * v \\ psi_{dot} = r \\ u_{dot} = c1 * v * r + c2 * u + c3 * (tu1 + tv1) \\ v_{dot} = c4 * u * r + c5 * v + c3 * (tu2 + tv2) \\ r_{dot} = c6 * u * v + c7 * r + c8 * (tr1 + tr2) \end{cases}$$

tu1, tu2, tr1, tv1, tv2, tr2 are intermediate variables that depend on which simulation we are in, which changes the engine configuration of the boat as well.

— in the front-rear engine case :
tu1, tu2, tr1 = p1*cos(theta1), p1*sin(theta2) , pont*p1*sin(theta1)/2,
tv1, tv2, tr2 = p2*cos(theta2), p2*sin(theta2) , - pont*p2*sin(theta2)/2

— in the differential push case :
tu1, tu2, tr1 = p1*cos(theta1), p1*sin(theta2) , pont*p1*sin(theta1)/2,
tv1, tv2, tr2 = p2*cos(theta2), p2*sin(theta2) , - pont*p2*sin(theta2)/2

these variables are set in order to make only one change of variable which allows not to change the state equation when changing the engine configuration.

From now on, we have two solution :

— Simply write a PID on p1, p2 by considering the boat speed. and another PID for the engine angle which, depending on the difference between the set course and the boat's heading and its own speed of rotation.

— we understand with these equations that there is no unique solution (4 unknowns for 3 equations), which is the case because our boat is over-actuated, which is supposed to make it more maneuverable which is normal, so in order to be able to solve the equation, because we know the speed udot and vdot through sensor, let's say that theta1 = - theta2, this means that if the boat wants to turn it will be turn the engines at opposite angles and then make it more handy.

$$u = A^{-1}(t) * (v - b(t)) \tag{2.1}$$

with v representing the objective (if v is null the objective is validated, - b(x) the singularity and A a simple coefficient matrix dependent on the state equation. Thus we obtain the vector u = (tu1 + tv1,tu2 + tv2, tr1,tr2) then we just have to do the change of variable to obtain the commands on the 4 corresponding motors u,v,theta.
Problem : in our case the differential delay on the speeds Vx, Vy are 1, while the heading has a differential delay of 2, this does not guarantee a perfect control. we can notice that on the python simulation, the boat prefers to follow first the set point in Vx, Vy and then he tries as best he can, to follow the course, below the differential matrix with u1,u2, theta in line and U, V, phi in column :

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix}$$

One thing we need to change simply by assuming that :

$$\begin{aligned} p1_{dot} &= F1 \\ p2_{dot} &= F2 \end{aligned} \tag{2.2}$$

so the differential delay should normally be catch up.

2.6 Simulation

At first we have done the simulation under python, but as we want to transfer everything to ROS, the boat dynamics will be recode on ROS. On this figure we can see the projection of the gps points in green marker but also a part of the velocity vector field (blue arrow) and the boat (white). Legend for Figure 1.3 :

— Yellow arrows : component on x and y of the setpoint vector (multiplied by the speedboat factor of the hand.py

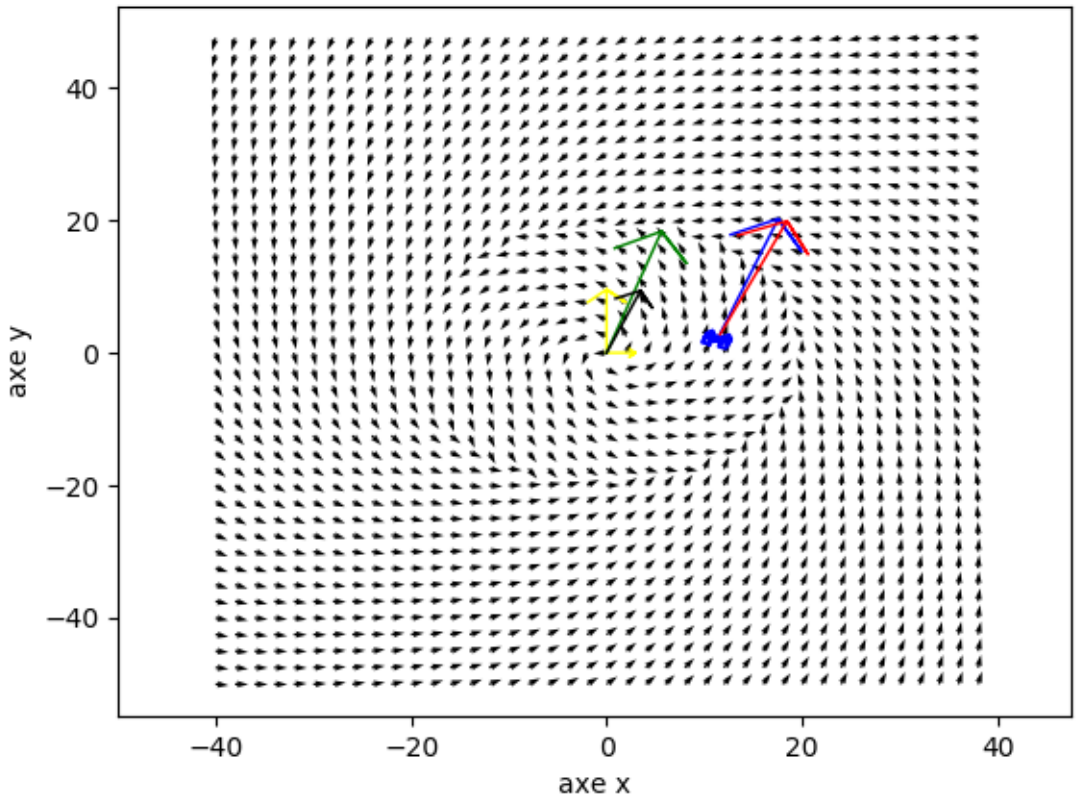

FIGURE 2.3 – Simulation on Python

— Green arrow : setpoint heading which is simply the result vector of the yellow arrows.
— Black arrow : boat's heading
— Red and blue arrow : engine thrust with angle and intensity

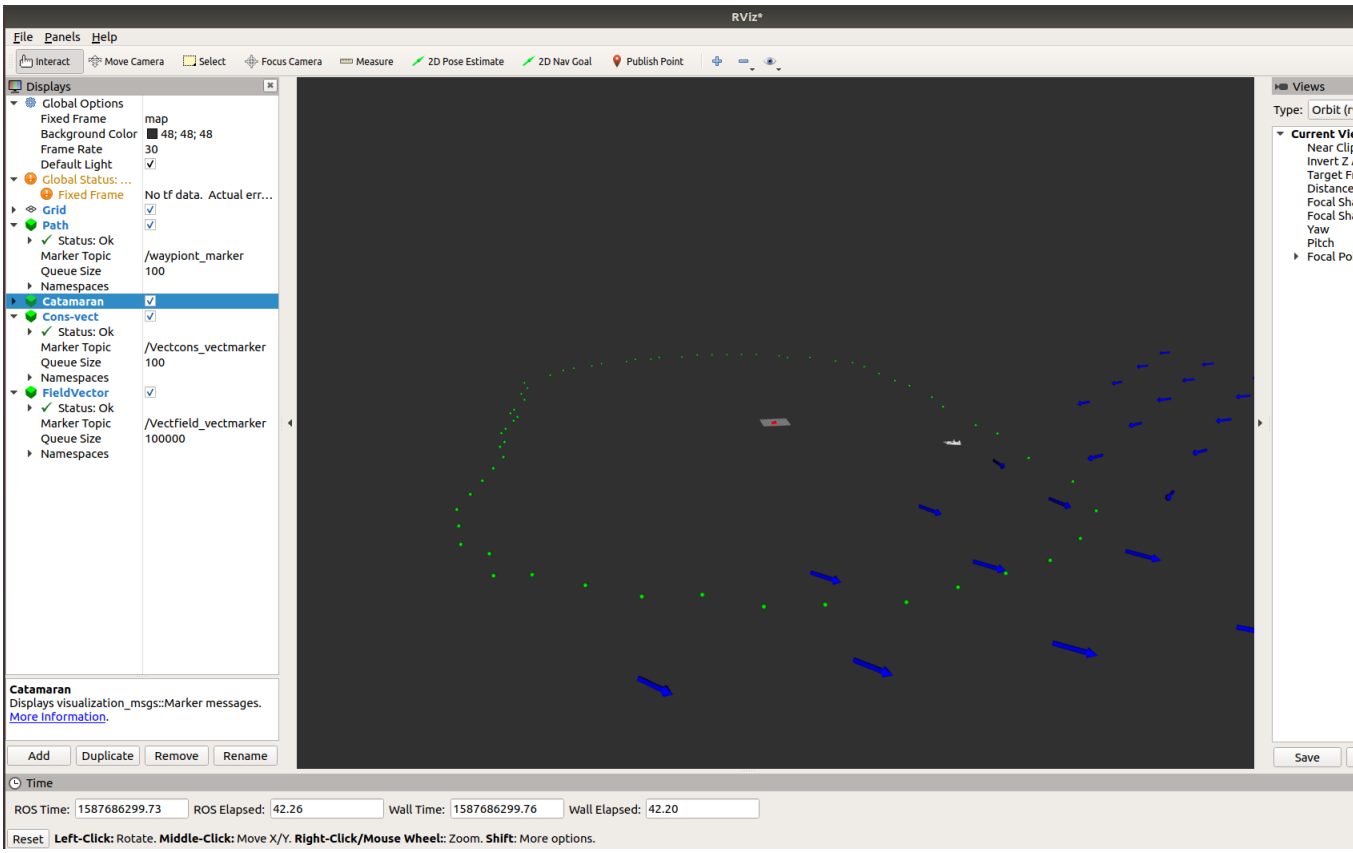

FIGURE 2.4 – workspaceCatamaran on RVIZ

Figure 1.4 :
— green point : waypoint (here they form the ENSTA stage)
— Blue arrow : Speed vector fields (partially visible)
— Red arrow : instruction for boat heading

2.7 Improvement

possible improvement. :

— reduce the complexity of the calculation of the velocity vector field (e.g. find the nearest point by operational search method but also for the search of the command, the file looks for the nearest point (x,y) for it associated with the command and again, finding the point by operational search would drastically reduce the calculation time 1
— make the application compatible on any machine and thus allow the execution and compilation with the whole imported library (so without having Qt creator).
— rewrites python code in C++ (in order to gain efficiency in compiling and creating the necessary files).

— Align all projections on the same projection (lambert, WGS, ...) because for the moment, more projections are used
— create the command for the static state of the boat
— make calculate once the display of the velocity vector field under VSWR, (for the moment the display of the velocity vector field forces its re-actualization which is useless (for the moment) but in any case it consumes too many resources for nothing and what prevents it from being displayed entirely.
— to make a seabed map appear on the application in order to be able to make a path planification in function.
— Perform Kalmann's filter on IMU, GPS, and compass data under ROS.