

MIDDLEWARE
DESIGN ET AUTONOMISATION D'UN KART
PROJET

KART

Gwendal PRISER
Paul-Antoine LE
TOLGUENEC
Quentin BRATEAU
Jules BERHAULT
Mamadou DEMBELE

April 15, 2020



Contents

1	Introduction	4
2	Modelisation	5
2.1	A kinematic model for the car	5
3	Functionnal Architecture	7
3.1	Introduction	7
4	Implementation Architecture	8
4.1	Introduction	8
4.2	Overview	8
5	Hardware	10
5.1	Introduction	10
5.2	Main Board	10
5.3	Sensors	10
5.4	Configuration	11
5.4.1	Pi Camera	11
5.4.2	Hardware PWM	11
6	Mechanical Architecture	13
6.1	Reasons of an Overhaul	13
6.1.1	Problems identified and changes to be expected	13
6.2	Front Wheel Steering	14
6.2.1	Front wheel slip	14
6.2.2	Ackermann steering geometry	14
6.3	Front Camera Support	15
6.3.1	Camera support structure	15
6.3.2	Annoying vibrations	16
6.4	Component Hosting Box	16
6.4.1	Additional space	16
6.4.2	Carved box	16
6.5	Expectations and Reality	17
7	Image Processing	18
7.1	Data logging	18
7.2	pre binarization treatment	18

7.3	binarization	18
7.4	post binarization treatment	18
7.5	find the center of the line	20
8	VREP Simulation	20
8.1	Explaination	20
8.2	Regulation	21
9	Localization	22
9.1	Problem in Localization	22
9.2	Robot status estimation	22
10	Discussions et Conclusion	23
11	Improvements	23
11.1	Prerequisites	23
11.2	Mission	24
11.3	GUI	24

List of Figures

1	State equation of the car	5
2	Differential delay graph of our car	6
3	Functionnal Architecture of this project	7
4	Implementation Architecture of this project	8
5	raspi-config utiliy on the Raspberry Pi	12
6	Steering system	15
7	Front camera support structure	15
8	Central component box	17
9	Expectations and reality	19
10	image processed	20
11	Graph node of the simulation	21
12	Explanation of the command : line in black, barycenter in red	21
13	State of yhe robot	23

List of Tables

1 Introduction

In the middleware module of UV 4.1, we are asked to implement the ROS middleware to drive a small car built in the previous semester. This small project consists in driving around an athletics track in an autonomous way. To carry out this project we have various sensors such as the Inertielle control unit which estimates the heading of the car and its acceleration, the camera, the gps which will allow us to estimate the position of the small car. We also have an on-board computer, in this case a raspberry.

Repo GitHub

<https://github.com/gwendalp/kart>

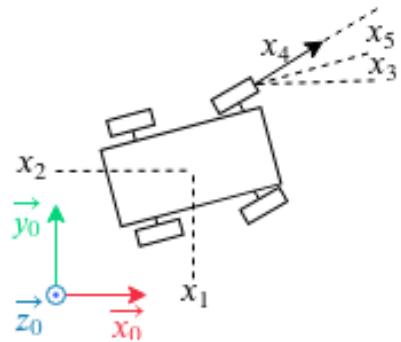
2 Modelisation

2.1 A kinematic model for the car

The first thing to take control of a system is to understand how it works. That's what we're going to do for this car. Indeed to control it we will have to establish these equations in order to elaborate the commands which will be used to lead the robot. **Figure 1** shows the cinematic equations of the car.

$$f(x, u) = \begin{cases} \dot{x}_1 = x_4 \cdot \cos(x_5) \cos(x_3) \\ \dot{x}_2 = x_4 \cdot \cos(x_5) \sin(x_3) \\ \dot{x}_3 = \frac{x_4 \sin(x_5)}{L} \\ \dot{x}_4 = u_1 \\ \dot{x}_5 = u_2 \end{cases}$$

(a) Equation of the car



(b) Diagram of the car

Figure 1: State equation of the car

In these equations x_1 and x_2 are respectively the x and y coordinates of the robot, x_3 is the heading of the robot, x_4 represent the speed of the rear motor and x_5 is the steering angle of the car controlled by the front servomotor. L is the distance between the front and the back driving shafts. In this system it is important to identify the variables of interest. These are the variables that we will want to control. In our case we will be interested in the speed of the car and its heading.

There is a useful tool to use when you want to understand how to run a system from state equations. The differential delay graph allows you to relate the inputs to the outputs of the car. Actually each new node represents an integration of the previous variable. There is also the link between inputs and outputs of our system. **Figure 2** is the differential delay graph for our

car. On this graph we represented the two variables of interest with a double circle.

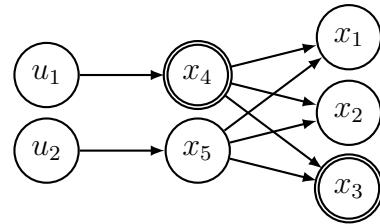


Figure 2: Differential delay graph of our car

3 Functionnal Architecture

3.1 Introduction

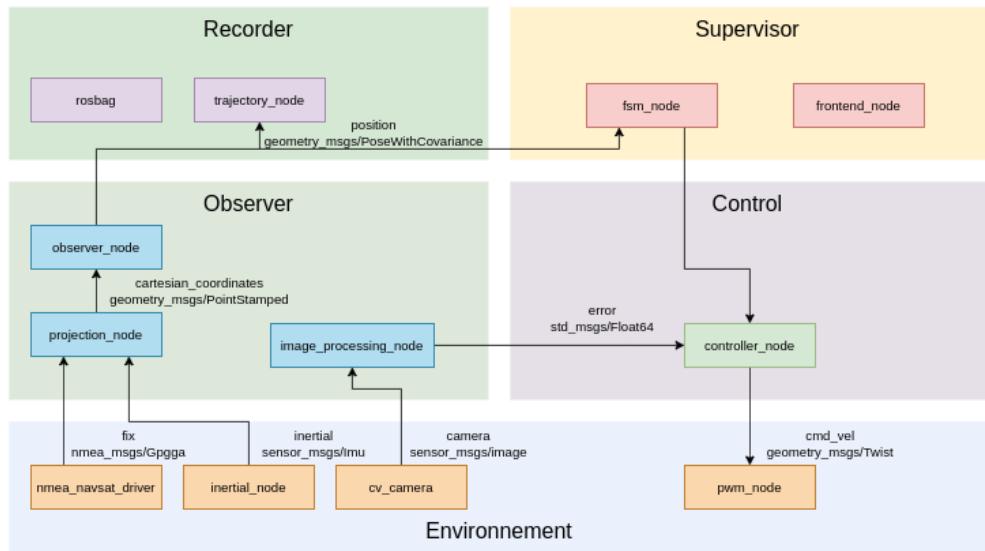


Figure 3: Functionnal Architecture of this project

This constitutes the functional specifications of the project and allows us to have criteria for the success of our project. **Figure 3** shows us the requirements diagram that follows the *SysML* standard for our system.

4 Implementation Architecture

4.1 Introduction

Divide and conquer. How better to describe Information Technology than by this concept? We therefore have to divide the code into different parts in order to better frame the problem and be able to solve it. In order to take control of the car, we decided to implement a control-observer architecture. This architecture divides the project into different parts and thus partitions the code. To implement this architecture, we can use a middleware like the *Robot Operating System* (ROS) which will allow us to simplify the communication between the different sectors through messages.

4.2 Overview

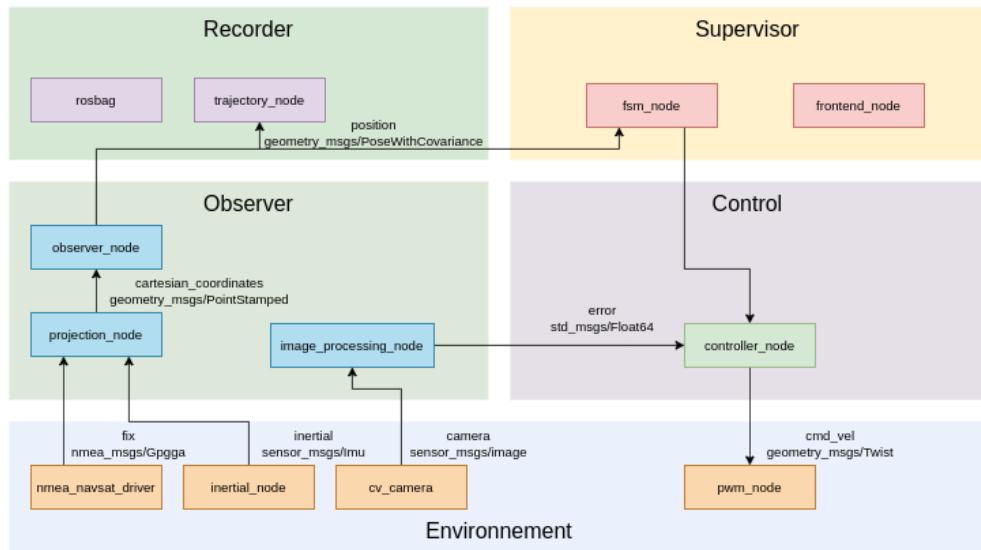


Figure 4: Implementation Architecture of this project

Figure 4 represents the implementation architecture of our project. In classical architectures, there are five main sections: Environment, Observer, Control, Recorder and Supervisor. They are then present in the architecture of our project.

Environment This section gathers all the low level nodes allowing the communication with the hardware. This entire section can be replaced by a simulation node that would publish the same messages in order to evaluate the rest of the functional architecture. In this project we have a global navigation satellite system (GNSS), an inertial unit and a camera. Finally we can control the speed of the rear engine and the position of the servomotor steering the car.

Observer This section is used to process data from the different sensors of the robot. In particular, it will be used here to perform image processing on the data from the camera. It will also allow to merge the data from the inertial unit and the GNSS in a Kalman filter to estimate more precisely the position of the car.

Control This section allow us to establish several control laws for our robot from the sensor data in order to drive our car's actuators. The Supervisor section will indicates the right control law on a case-by-case basis.

Recorder This section records the various robot data in order to be able to replay the data, particularly through rosbags, and it records the trajectory of the robot to feed the finite state machine.

Supervisor This section consists mainly of the finite state machine that controls the mission. It will determine the right command law to control the robot according to what the robot can perceive from its environment. There is also a front-end server that allows us to have a feedback, a graphical user interface to know the state of the robot.

5 Hardware

5.1 Introduction

In this section we will describe the hardware configuration of this project. The car is composed of a set of sensors, an interface board, and a board allowing software processing. The assembly must meet the constraints of the embedded systems, i.e. constraints of size, weight and power consumption.

5.2 Main Board

For this project, we decided to use a *Raspberry Pi 3B+* as main board. It will allow us to connect all the sensors necessary for the operation of the car, but also to carry out the software processing implemented in the car.

On this *Raspberry Pi* we will need to install an operating system in order to run our programs. Furthermore, the connectivity of this board is suited to our problem since we have USB ports as well as access to the GPIO pins that allow us to control several devices.

5.3 Sensors

We have at our service for this project 3 different sensors that will allow the robot to acquire information about its state: an *Inertial Unit*, a *Global Navigation Satellite System* (GNSS) and a *Camera* interfaced to the dedicated CSI port.

Actually, for our problem we found that an accuracy of 1 meter for the *GNSS* is too large because the car need to run in a 0.8 meter wide racing lane, following a line. This sensors is also not able to know if the car position is correct.

For the *Inertial Unit*, we found that this sensor is too noisy to give us any usefull informations about the state of our car. For instance the consecutive integration of the acceleration in order to get the speed and the position of the car leads to an important drift effect on our data. So this sensor is not currently able to give any correct informations about the state of the car. Moreover, the acceleration of the car could be quite good after filtering if we only needed it. In our case the only usefull information is to know the position of our car in relation to the line.

That's why we decided to focus our attention on the camera. Because we are using a *Raspberry Pi 3B+*, the camera we have choosen is the official camera which can be plused on the dedicated port on the board. This sensor is perfectly suited to our problem, because with an appropriated image processing we will be able to detect the line and to correct the car trajectory.

5.4 Configuration

Now we will explain how we configured our sensors in our project, to let them comunicate with the software and with the car.

5.4.1 Pi Camera

The configuration of the camera on the raspberry pi is relatively simple. We used the *raspi-config* utility to configure the camera. That's how we set up the camera on the Raspberry Pi.

5.4.2 Hardware PWM

On Raspberry Pi board there is a lot of way to generate pwm signals. The most of the time, these methods are software based and so they are not accurate. With a lot of searches, we found a website who speak about the raspberry pi's hardware pwm signals. There is apparently an hardware pwm generator used by the bord to generate sounds. It's better to use hardware generated pwm, because if the processor has a slow down and the interrupt is not correctly handled, the pwm duty cycle will not be very accurate and

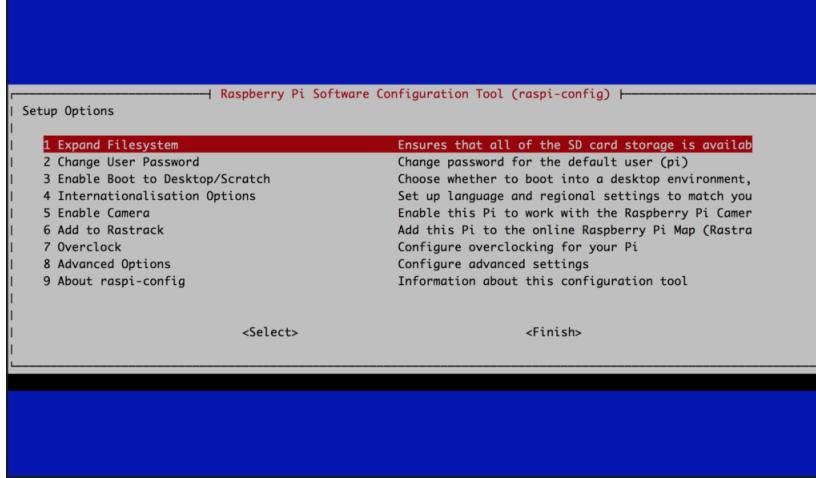


Figure 5: raspi-config utiliy on the Raspberry Pi

the car will not be able for instance to follow a straight line, because the bearing of the car is controled by a servomotor with a pwm signal.

So we decided to use this tutorial : [1], which explain us how to setup pwm signals on the Raspberry Pi, and how to correctly configure the files to have a standard pwm signal which is generated. Then we need to give the rights to users for reading and writing in these files. All these bash command are in *init_pwm.sh*.

Then we have to add some automation. So we created a *crontab* rule. That will automatically create all the required files and allow the permissions to every users. We just have to write in the file *duty_cycle* a value between 1.000.000 and 2.000.000, and the Raspberry Pi will read and adjust pwm signals in real time.

Last but not least, we setup an autologin in order to open a session automatically when the Raspberry Pi boot. That's very usefull in order to launch our programms easily on boot and without any keyboard, mouse or monitor.

6 Mechanical Architecture

6.1 Reasons of an Overhaul

The original architecture of the vehicle was already a good basis for the realisation of a line tracking car. However, manual manoeuvring tests by remote control have revealed some driving faults. Misconduct that could be embarrassing in an autonomous driving mission. Given that our robot will certainly not be as adaptive as a human in its driving, it is interesting to ease the maneuvers and correct some mobility deficiencies.

6.1.1 Problems identified and changes to be expected

Some of the shortcomings noted are listed as follows:

- Loss of control due to high front wheel slippage during high speed turns.
- In case of sharp turn, locking of the inner wheel on the bend caused by the central component box.
- Lack of firmness of the front wheel guiding created by a large backlash.
- Uncontrolled spinning of the rear wheels when acceleration from a standstill or deceleration from high speeds.

And some others component have to be implemented to the initial structure in order to be an optimal support for autonomous driving such as:

- The installation of a camera to see and locate the line to follow.
- Hosting a Raspberry Pi card to manage the control computations.
- Keep space for the rest of the essential components such as the battery, ESC, cables, etc.

6.2 Front Wheel Steering

6.2.1 Front wheel slip

One of the major problems that needs to be corrected, a problem that could impact autonomous driving, is the significant slipping of the front wheels when the steering angle is too large at high speed.

To correct this defect, the best solution would be to improve road handling by increasing the car's grip. To do this, the wheels would have to be changed from smooth tires to studded wheels. However, considering the stock of parts available at ENSTA Bretagne, unless you make an order which would take some time, we did not have better wheels than the ones proposed.

6.2.2 Ackermann steering geometry

Nevertheless, we have come up with another solution inspired by normal cars. The principle of this solution comes from the geometry, especially the steering geometry of Ackermann.

The intention is to avoid the need for tyres to slip sideways when following the path around a curve. The geometrical solution to this is for all wheels to have their axles arranged as radii of circles with a common centre point. As the rear wheels are fixed, this centre point must be on a line extended from the rear axle. Intersecting the axes of the front wheels on this line as well requires that the inside front wheel be turned, when steering, through a greater angle than the outside wheel.

A simple approximation to perfect Ackermann steering geometry may be generated by moving the steering pivot points inward so as to lie on a line drawn between the steering kingpins and the centre of the rear axle.

We have therefore redesigned the pivot points to respect this geometry. We also took advantage of this to improve the bearing embedding system that connects the bearing to the wheel so that the steering wheels are held more firmly.

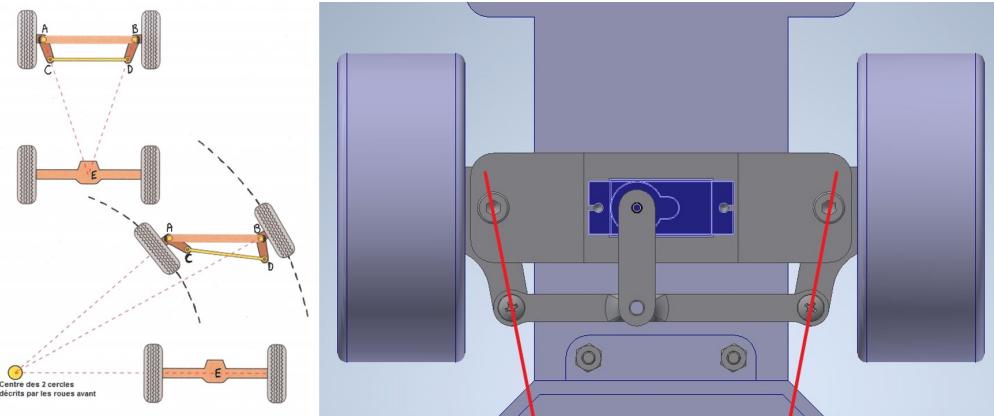


Figure 6: Steering system

6.3 Front Camera Support

6.3.1 Camera support structure

We then had to implement the camera that would allow our robot car to have eyes on the road.

In order to position it correctly, the camera had to be able to stand back enough on the road in front of it and the angle of view had to be horizontal enough to see far away but with a high enough incidence to identify the line without being hindered by the sun's reflections.

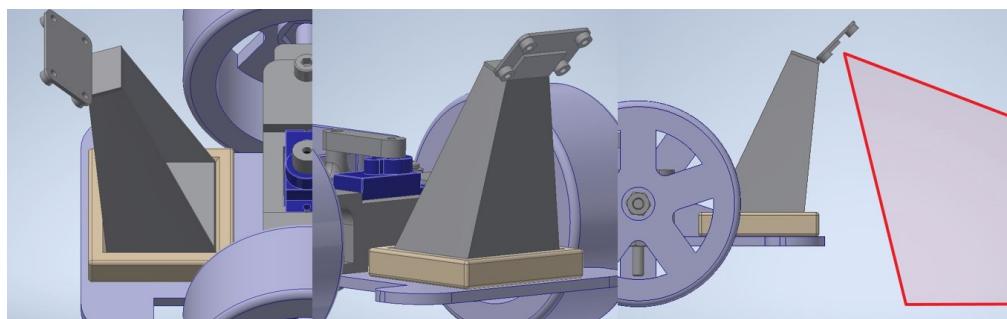


Figure 7: Front camera support structure

We have therefore designed a structure that raises the camera a few centimetres above the chassis and tilts it at 45 degrees towards the ground.

6.3.2 Annoying vibrations

Our car will be driven in a natural environment and the ground will probably not be totally smooth but slightly bumpy and gravelly. The chassis will be subjected to many vibrations and at the same time, the camera to which it is connected will also be subjected to them, which can be quite annoying afterwards in the image processing.

We therefore had to find a solution to try to reduce these disturbances. One of the simple mechanical solutions is to dampen the vibrations in the connection between the chassis and the camera by using for example a silent-block. As a silentblock, a soft synthetic foam seems to be a good choice to counteract these vibrations. (See Figure 5 for more details)

6.4 Component Hosting Box

6.4.1 Additional space

In order to host the necessary additional electronics we needed to increase the space to host the components. We redesigned the box by increasing its width. In its new dimensions, the box offers twice as much space without unbalancing the overall structure.

6.4.2 Carved box

In the process of creating a new box, we took the opportunity to correct one of the problems caused by the front corners that blocked the steering wheels when they were fully steered. So we took the care to add two notches in the place of these corners to allow the wheels to move without hitting them. (See Figure 6 for more details)

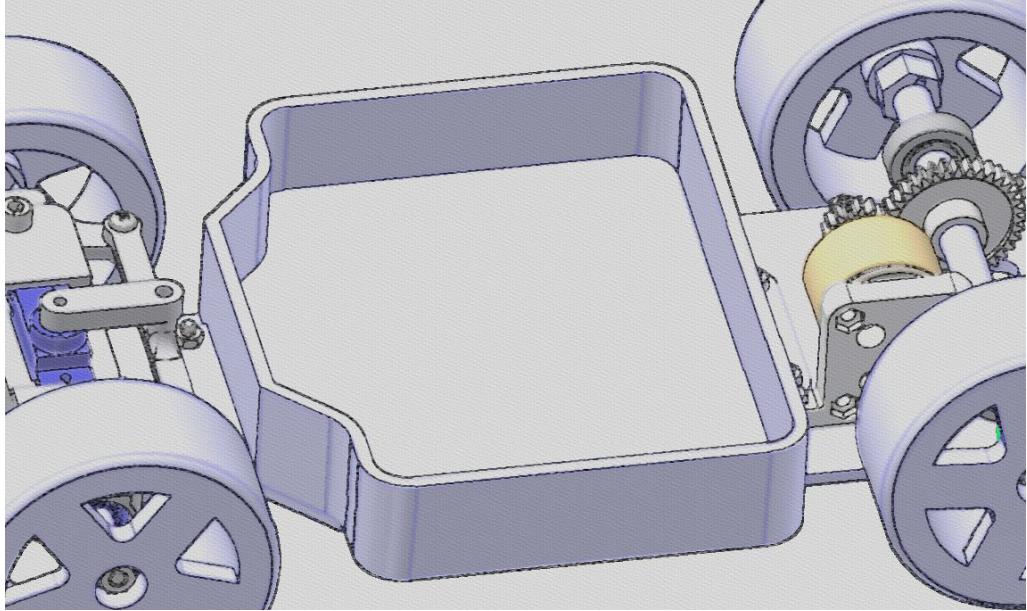


Figure 8: Central component box

6.5 Expectations and Reality

As you may probably know, there is a difference between expectations and the actual reality.

The plans were good, but an unprecedented event came to modify them, as a consequence of the proliferation of covid-2019, the school had to close its doors. Deprived of manufacturing means and having only one prototype for the group, we had to stick to what was already done. With DIY, hot glue and a bit of common sense we were able to come up with a modest but acceptable structure.

In the final model, the proposed solutions include Ackermann's steering system and the system for embedding the bearings connected to the steering wheels. The front camera support had to be totally modified but the intentions are the same even though there is no damping system. And the housing box has remained the same but has been doubled on a second floor

to contain the electronic components. By limiting the rotational movement of the steering wheels, we can ensure that the front wheels do not get stuck. (See Figure 7 for more details)

Although the final structure of our car does not have all the improvement imagined, it still in acceptable condition for autonomous driving in an environment like a traditional athletics track.

7 Image Processing

7.1 Data logging

For the first part of this image processing, we had to collect images. So first we went into the environment in which the robot was going to evolve.

7.2 pre binarization treatment

In this part, we first had to perform a pre-binarization treatment in order to reduce post-binarization noise. So we used a Gaussian filter to blur the image.

7.3 binarization

Since the line we wanted to mark is white. An effective treatment is simply to switch to grey level. So for binarization, we switch the image to a grey level and threshold for a grey level that we have determined empirically.

7.4 post binarization treatment

In this part we performed a morphological treatment. There was still a lot of noise after binarization. So we made an opening. With a kernel in the shape of a rectangle (since it was the most efficient for this treatment).

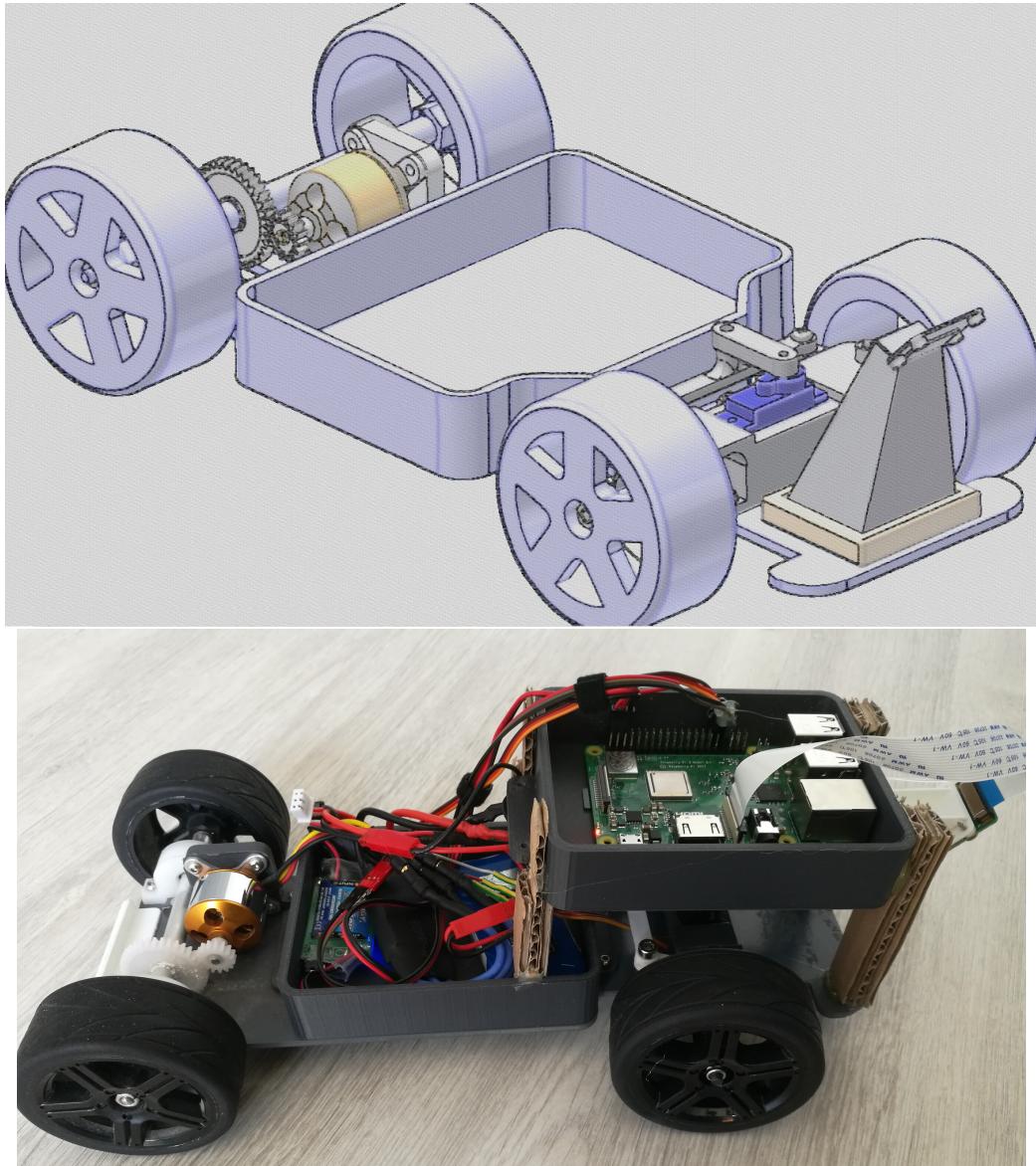


Figure 9: Expectations and reality

At the end of this treatment we obtain a well defined line which crosses the screen.

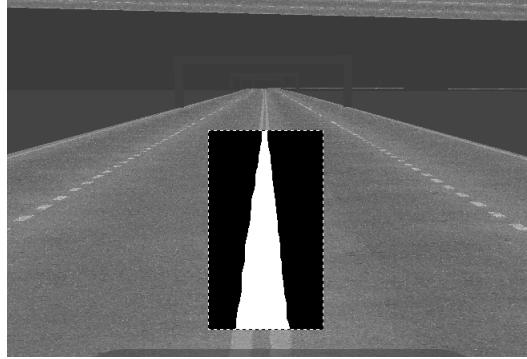


Figure 10: image processed

7.5 find the center of the line

In this last part, the contours are marked using a gradient method. Then the contours are sorted from the smallest to the largest. We recover the largest contour. And we recover the coordinates of the barycentre of the contour. Then the error is the difference between the center of the image and the coordinates of the pixel.

8 VREP Simulation

8.1 Explanation

Based on the practical work made with Benoit Zerr, we made a realistic track with a line in the middle [2]. The simulated in VREP car is equiped with a camera. Our VREP simulator is interfaced with ROS through the piece of LUA code :

As it is explained on the figure 11. The VREP camera send a through a topic `image`, a picture of the front of the car. Then the `camera_node` subscriber process the image with OpenCV and return the position of the barycenter of the line (red in the figure 12), then it compute an error, we want the barycenter of line in the middle of the windows, as it's shown on figure 12. This error is sent through a topic `error`, the controller is suscribed to this topic, it generate a command thanks to a PD command law which

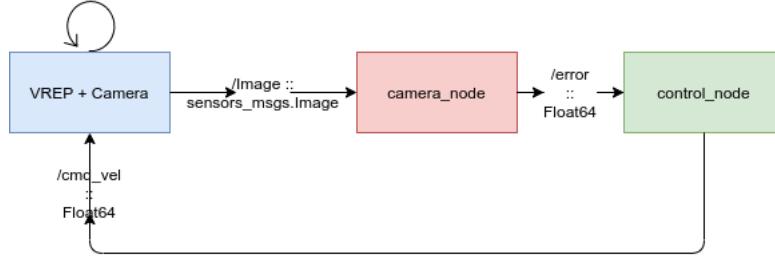


Figure 11: Graph node of the simulation

is sent to VREP. And now the car can loop forever, and achieve the perfect run in 1'15" ! 🚗🏁⌚

8.2 Regulation

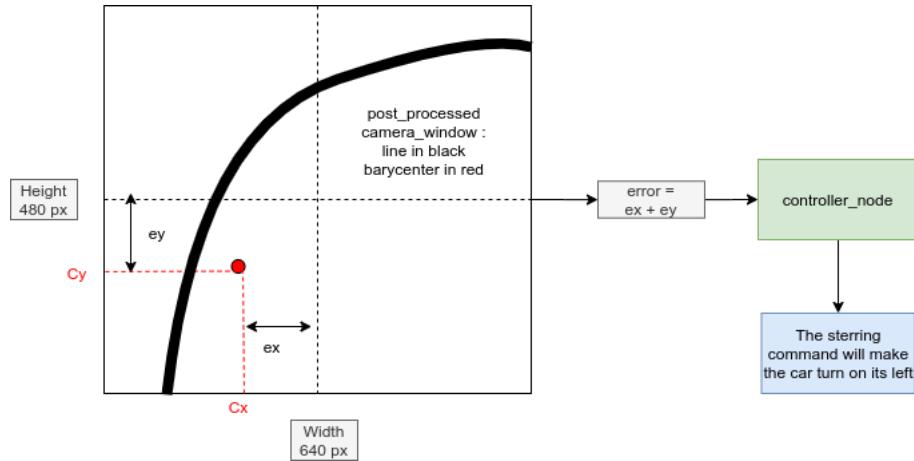


Figure 12: Explanation of the command : line in black, barycenter in red

We implemented a PD regulation. Let's call (c_x, c_y) , the barycenter of white line, the car is required to stay in the middle of the line. Therefor we compute an error : $e = (width/2 - c_x) + (height/2 - c_y)$ and then the command u equals : $u = K_p e + K_d \frac{de}{dt}$. This command is computed in the controller and sent to vrep.

The result in video : ¹

9 Localization

9.1 Problem in Localization

In this project we decided to realize the challenge by a visual servoing. That is to say a line tracking by the camera. This type of servo control only works if the robot has at each moment the line to follow in its field of vision. This kind of problem can happen when the robot misses the turn. If the robot ever loses sight of this line, it is obvious that the challenge will not be successful.

To compensate for this unforeseen situation, we have implemented a control law that will be used as soon as the robot loses the line of sight sound. This control law requires knowledge of the robot's state.

9.2 Robot status estimation

To compensate for this unforeseen situation, we have implemented a control law that will be used as soon as the robot loses the line of sight sound. This control law requires knowledge of the robot's state.

Where (x,y) is the position of the robot, θ is its heading, v its speed and δ the angle of the front wheels. To be able to apply the Kalman filter we need a linear state representation of the robot. To make these equations linear, we have considered the angle θ of the robot as an input (since it is given by the inertial unit). This removes equation 3. Then we linearized the equation

¹https://www.youtube.com/watch?time_continue=10&v=_vIXo1TvG0w&feature=emb_logo 

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\delta} \end{pmatrix} = \begin{pmatrix} v \cos \delta \cos \theta \\ v \cos \delta \sin \theta \\ v \sin \delta \\ u_1 \\ u_2 \end{pmatrix}$$

Figure 13: State of yhe robot

into \hat{x} (\hat{x} being the new state to be determined). \hat{x} consists of x, y, θ, δ . The file available on our github shows the function of this Kalman filter.

10 Discussions et Conclusion

To conclude, we think that we already have a strong basis for this project, particularly with the ROS structure which is already setup. Then we have to perform test with the real system because the simulation is correctly working and a good proof of concept, but we know too that simulation and reality are always different. We have to do some improvements on our project too, like adding an mission controller in order to control the robot in some cases where we couldn't use the line following command, for instance when the camera is not seeing any lines, and we ha to add a GUI in order to show the kart state like his position, is speed and the circuit.

11 Improvements

11.1 Prerequisites

This part will be implemented in the future and are for now only some good ideas we have for this project. We are first focused on the achievement of the simple control of this car.

11.2 Mission

We have establish a simple command for our kart based on image processing. This command law is quite easy to setup and fully functionnal as we could see. However this command is efficient only if the camera is able to see a line. But actually the car could be unable to see a line, particularly when the car will accelerate in order to run a lap as quick as possible. Our strategy on this topic is to run a first lap very slow and to drop virtual gps tags on a map while the car is following the line. So after a lap we will get a whole map of the circuit and at any given times the car will have to determine if a line is available on the camera, and if the car could follow this line, else the car will have to reach this line again with the help of the gps tags. Then after the first lap, the car will be able to accelerate and for instance some times lost the line because it will be able to reach again the circuit with the gps tags.

11.3 GUI

We found quit interresting to have a beautifull Graphical User Interface for our car. We are thinking about a frontend solution in order to show dynamical parameters such as acceleration and seep of the car, and a map to show the position with the classical circle to represent the incertainty of this position. Moreover it could be interresting to have the camera output available on this GUI. The easiest way to display a map and some gnss coordinates on python is to use the *folium* package based on the *OpenStreetMap* map. We are now working on a bind between *ROS*, *folium* and the web framework *Django* to sum up these informations on a state webpage. However, we don't own any GNSS or Inertial Units to prepare hardware nodes. That is also not our prioriy, but it can be a very usefull tool.

References

- [1] Pi Zero W Rover Setup | Disconnected Systems. <https://disconnected.systems/blog/pi-zero-w-rover-setup/#moving-the-robot>.

- [2] Benoît Zerr | TP VREP. <https://www.ensta-bretagne.fr/zerr/dokuwiki/doku.php?id=vrep:create-rc-car-robot>.