MIDDLEWARE

Autonomous Car

Project

KART

Gwendal Priser
Paul-Antoine Le Tolguenec
Mamadou Dembele
Quentin Brateau
Jules Berhault

April 23, 2020

Contents

List of Figures

List of Tables

1 Introduction

In the middleware module of UV 4.1, we learned one of the most used software architectures in robotics: Robot operating System. In this module we learned the basics of ROS and its importance in a robotics project. In order to apply the knowledge acquired in ROS to a concrete project, we were asked to carry out the KART project. This small project consists in driving around an athletics track in an autonomous way.

The Kart project consists in making a small car autonomous. In addition to this, the car must be able to do a complete tour of an athletics track. To carry out this project, we have various sensors such as an inertial measurement unit that estimates the direction and acceleration of the car, a camera, a GPS that 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 Functionnal Architecture

2.1 Introduction

The functionnal architecture constitutes the specifications of the project and allows us to have criteria for the success of our project. **Figure** 1 shows us the requirements diagram that follows the *SysML* standard for this system.

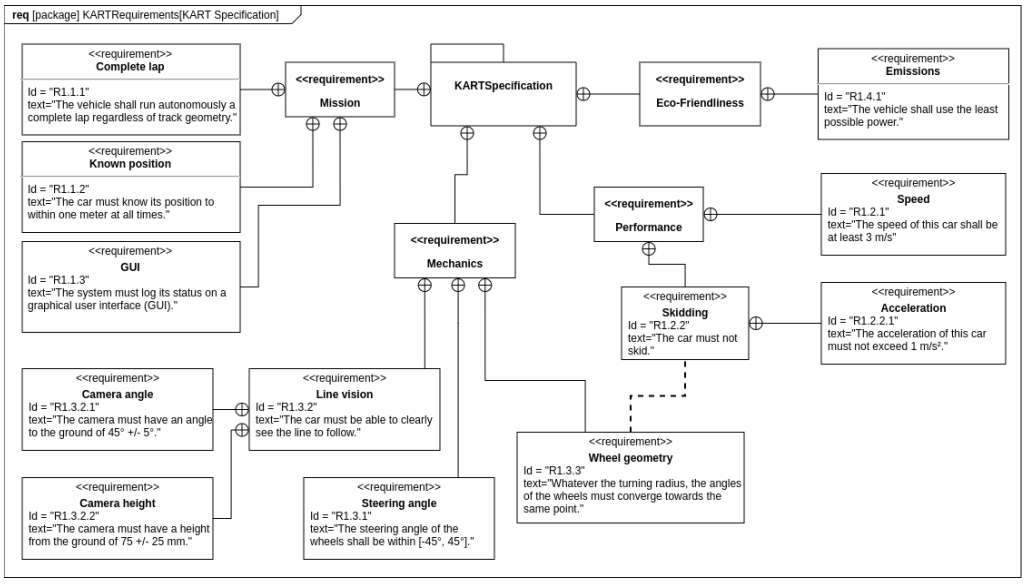


Figure 1: Functionnal Architecture of this project

The requirements diagram in **Figure** 1 shows us the requirements we want for this system. The requirements are divided into several categories.

First there is a mission requirement that the robot must follow. It must therefore know its position and be able to communicate it to the user through a graphical user interface. In addition, the car must perform laps around the track regardless of its shape.

Then we have requirements in terms of the mechanics and design of the robot. The car must not skid on the track under any circumstances, as this would lead to a loss of control of the track. In addition, there is a requirement for the camera to be positioned and oriented so that it can observe the line in the best possible conditions.

Then there are performance requirements, including a minimum speed to be respected, as the goal of the mission remains to complete laps of the track as quickly as possible. There is also a requirement for maximum acceleration, which will allow the car to remain in control of the car while avoiding loss of grip on the track.

Finally, as the system is an on-board system, it operates on battery power and therefore it must meet a criterion of resonant battery usage. It should be noted that by limiting the car's acceleration, the energy balance is much better and the battery is saved. This allows the system to work longer and in better conditions.

3 Implementation Architecture

3.1 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.

3.2 Overview

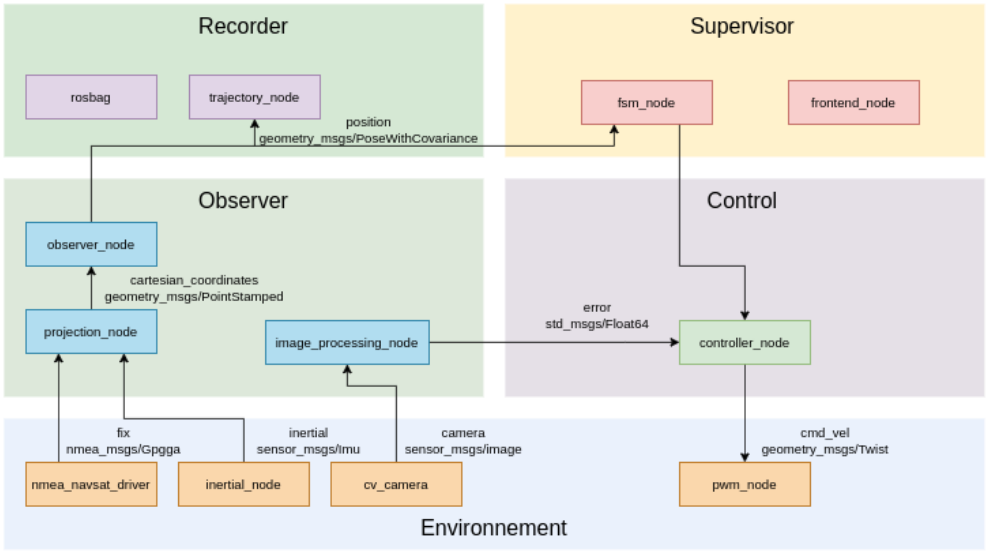


Figure 2: Implementation Architecture of this project

Figure 2 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.

We have also implemented a user interface for providing feedback. Indeed, we thought it was important for the user to be able to have an idea of the different parameters of the car, such as its position, its acceleration or the image the car sees.

So we used the Flask Framework and made a link with ROS within a node. Flask allows to set up a web server on the car. An http request can be sent from a browser and allows access to this information. We also used

the Folium library in python to provide a map to see the trajectory of the car as shown in **Figure** 3.

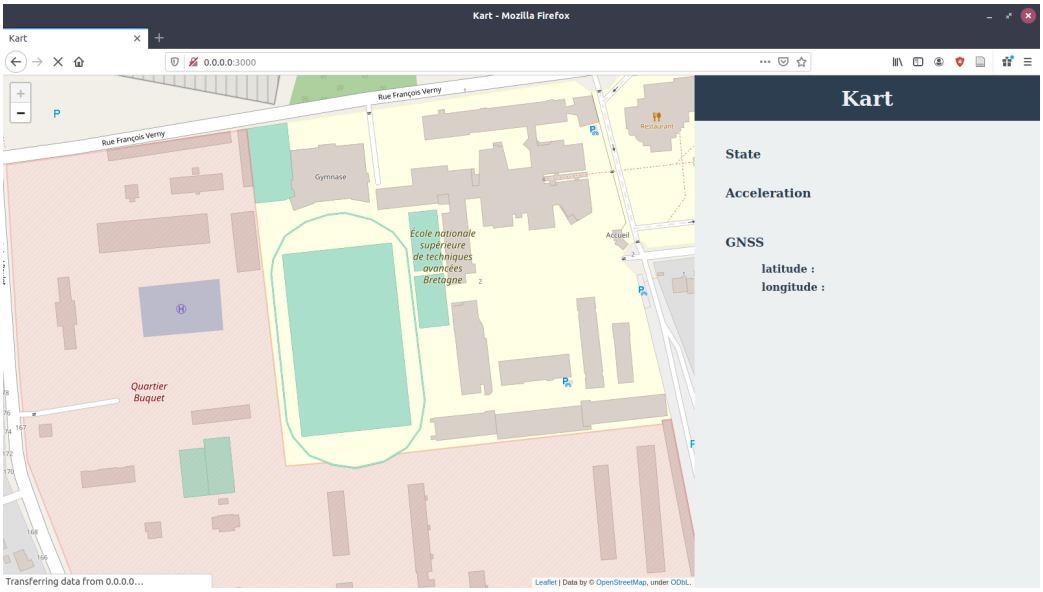


Figure 3: Flask webserver for feedback

The problem is that currently we don't have any GPS nor inertial unit. So we can't have this live feedback, but it is still possible to provide fake data which allowed us to validate the operation of this web server.

To conclude, we can notice the presence of this architecture in two loops. We have a small loop formed by the camera, image processing, controller and pwm nodes, and a much larger loop involving the gps and the finite state machine. This relates well the will to want to achieve the global mission by taking into account all the cases, to always have a command to give to the car.

4 Image Processing

To carry out the image processing we used very simple theoretical notions. The library we used is open cv. The figure below summarizes our image processing.

Figure 4: image processed

4.1 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.

4.2 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.

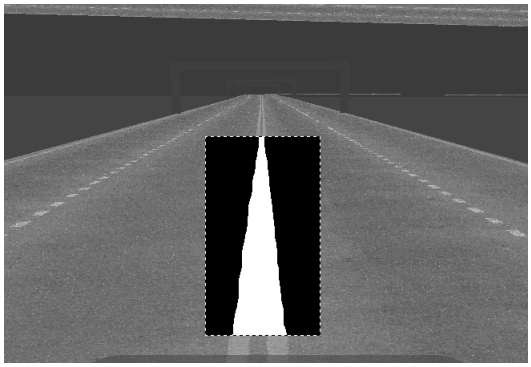


Figure 5: image processed

4.3 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.

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

4.5 Barycenter 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. The problem with this method is that with the barycentre we have a regulation on a point in the centre of the image. And therefore, we don't use all the data we have. So we decided to use the

highest points of the contour. Indeed, the position of the center of the line at the highest point of the image gives us an idea of the evolution of the road at the moment. From now on we use the horizon. It allows us to get information further into the "future". And so it allows for better regulation and therefore to go faster.

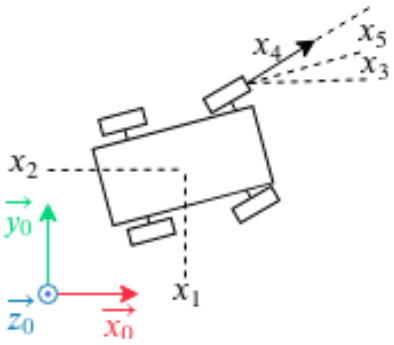
5 Controller

5.1 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** 6 shows the cinematic equations of the car.

$$f(x,u) = \begin{cases} \dot{x_1} = x_4.cos(x_5)cos(x_3) \\ \dot{x_2} = x_4.cos(x_5)sin(x_3) \\ \dot{x_3} = \frac{x_4 sin(x_5)}{L} \\ \dot{x_4} = u_1 \\ \dot{x_5} = u_2 \end{cases}$$

(a) Equation of the car



(b) Diagram of the car

Figure 6: 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 usefully 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** 7 is the differential delay graph for our car. On this graph we represented the two variables of interest with a double circle.

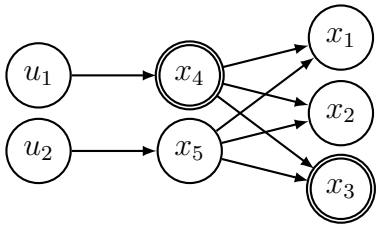


Figure 7: Differential delay graph of our car

As we can see on the differential delay graph shown on **Figure** 7, there is only one integration between the inputs and the x_4 output against a minimum of two integrations between the inputs and the x_3 output. We will therefore be able to act faster on the speed of the car than on its heading. Moreover we notice that to change the orientation of the car, we need both inputs. Indeed, the orientation of the front direction has no effect on the orientation of the car if it is motionless.

5.2 Simple law of control

In the case of a system that was originally built to be driven by a human, reference [**?**] tells us that it is possible to develop simple controls that would imitate the behaviour of a human pilot. Indeed, by following thetwo principles stated below, it is possible to take control of the car in a very simplistic but at least functional way.

- **First principles** The speed of the car must be constant and not null.
- **Second principles** The further the car is from the target, the more the car will turn.

With the first principle, we ensure that the car will run at all times, and therefore that the action on the steering will allow the car's heading to

be adjusted. The second principle simply indicates that the further the car moves to the right of the line, the more it will turn to the left and vice versa. Thus the car will tend to return closer to the line.

According to reference 2, it is necessary to take the arctangent of the error in order to have a saturation effect on the control. The error here is of course the distance between the center of the car and the center of the line. We have chosen the arctangent shown in **Figure** 8. With this function, we are sure that the order law will not diverge, even if the line is far from the center of the picture.

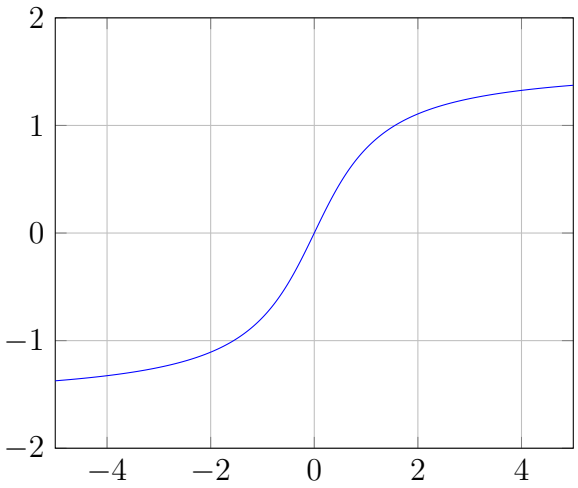


Figure 8: Arctan control saturation for car steering

In conclusion, noting e the error and v_0 the desired speed which will be constant, this simple control law will come from the equations 1.

$$\begin{cases} u_1 = v_0 \\ u_2 = arctan(e) \end{cases} \tag{1}$$

6 Hardware

6.1 Introduction

This section will present the hardware architecture of this project. The assembly must meet the constraints of the embedded systems, i.e. constraints of size, weight and power consumption. As shown in figure 1, the car is based on the Raspberry Pi 3B+ board and has a rear engine managed by an ESC and a servo motor to control the steering at the front. Finally, as far as sensors are concerned, we have at our disposition an *Inertial Unit*, a *Global Navigation Satellite System* (GNSS) and a *Camera* interfaced to the dedicated CSI port.

6.2 Main Board

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.

6.3 Sensors

We have at our service for this project 3 different sensors that will allow the robot to acquire information about its state:

GNSS Provides access to the latitude and longitude of the car. In combination with a Kalman filter, it satisfies the need to know the position of the car. It could also be useful in cases where the car loses track of the line, in order to come back to it.

Inertial Unit Allows access to the car's acceleration. This is useful to limit the car's acceleration and thus prevent the car from skidding in its path. However, we do not have it available at the moment, so we have not been able to implement it within this system.

Camera Because we are using a *Raspberry Pi 3B+*, the camera we have choosen is the official camera which can be pluged 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.The setting is quite simple and is done via the *raspi-config* menu. Moreover the ROS hardware node *cv_camera* that we found interfaces automatically with the camera connected to the dedicated CSI port. So we just have to get the image on a topic to do the image processing.

6.4 Actuators

In order to manage the direction and speed of the car, we need to be able to generate PWM signals. The first signal will control the ESC that manages the rear engine. The second signal to be generated will allow to manage the steering of the car via the control of the front servomotor.

On a *Raspberry Pi* board there is several way to generate pwm signals. The most of the time, these methods are software-based. It's better to use hardware generated pwm, because if there is a high CPU usage, interrupts will not be correctly handled, the pwm duty cycle will not be very accurate and 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.

According to this article [1], there are two hardware PWM pins on a *Raspberry Pi 3B+*. Some automation was required to set up the file system to allow the use of hardware PWM. So we had to use crontabs to keep the installation up and running. After this installation the pwm is operational and allows us to fully control the rear engine and servo motor of the car.

7 VREP Simulation

7.1 Explaination

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 :

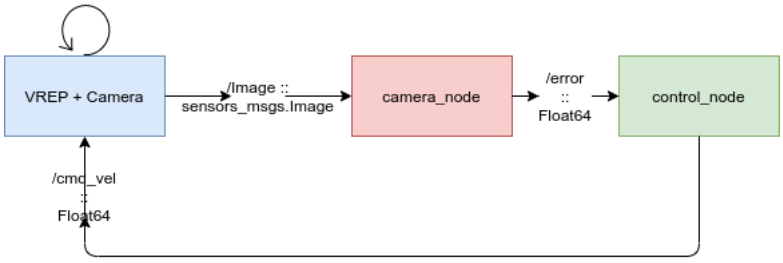


Figure 9: Graph node of the simulation

As it is explained on the figure 9. 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 10), then it compute an error, we want the barycenter of line in the middle of the windows, as it's shown on figure 10. 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 is sent to VREP. And now the car can loop forever, and achieve the perfect run in 1'15" !

7.2 Regulation

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]

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

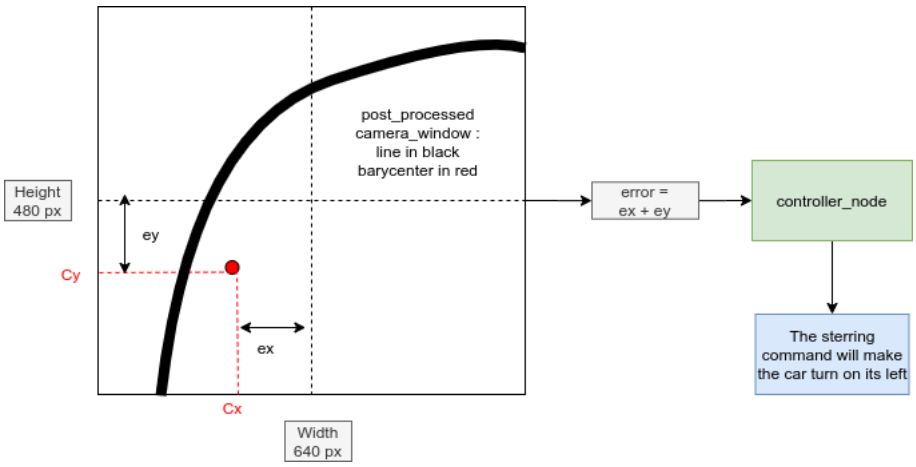


Figure 10: Explanation of the command : line in black, barycenter in red

8 Localization

8.1 Problem in Localization

In this project, we decided to realize the challenge through visual servoing. That is to say a line tracking by the camera. This type of servoing only works if the robot has at all times the line to follow in its field of vision. But this is not possible since it will very often happen that the robot loses sight of the line. Especially during a turn. So if our car loses sight of the line, we'll have to get it back on the track.

8.2 Robot status estimation

To compensate for this very likely situation, we have a control law in place that will be used as soon as the robot loses the line. This law of control requires knowledge of the robot's condition.

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

Figure 11: State of the robot

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

9 Mechanical Architecture

9.1 Reasons of an Update

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. Defects that could be disruptive 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.

9.1.1 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.

Since we're starting from the structure of a radio-controlled car not designed to host our computing system (RasPi3, camera, etc.), some components have to be implemented and the structure has to be reworked 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 3 card to manage the control computations.
- Keep space for the rest of the essential components such as the battery, ESC, cables, etc.

9.2 Front Wheel Steering

9.2.1 Front wheel slip

One of the major problems that needs to be solved, 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 parts stock available at ENSTA Bretagne, unless we make an order which would take some time, we did not have better wheels than the ones proposed.

9.2.2 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 tyres to slip sideways when following a curved path. The geometrical solution to this is for all wheels to have their axles

arranged as radii of circles with a common centre point. (See 13 for more details) 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 to be turned, when steering, through a greater angle than the outside wheel.

A simple approximation to improve 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.

9.3 Front Camera Support

9.3.1 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 on the road. This results in false identification of the lines and then adds errors in the guidance.

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.

9.3.2 Vibrations issue

Our car will be driven in a natural environment and the ground will probably be slightly bumpy and gravelly. The chassis will be subjected to many vibrations and at the same time, the camera may be subject to vibrations

transmitted by the chassis. These can degrade the performance of the image processing algorithms.

One of the simple mechanical solutions to reduce these disturbances is to dampen the vibrations in the connection between the chassis and the camera by using for example a silentblock. We used a soft synthetic foam as a silentblock to counteract these vibrations. (See 14 for more details)

9.4 Component Hosting Box

9.4.1 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.

9.4.2 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 15 for more details)

9.5 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.

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.

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 16 for more details)

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

10 Conclusion

In conclusion, we have first established the requirements of this system. The main objectives are to ensure that the car performs laps of the track, regardless of its geometry, in autonomy and as quickly as possible. Then the use of ROS allowed us to separate the functionalities in different nodes in order to facilitate the development of the system. This allowed us to divide the work but also to be more efficient in the development of the software architecture of the car. Then we made a simulator that allowed us to validate the different requirements of this system. It was very useful in the current context which did not allow us to test the car in a real environment. We were able to validate a minimum speed criterion of $3m.s^{-1}$, because we were able to push the speed on the simulation up to $6m.s^{-1}$. Then the range was checked as the car was able to finish the lap in simulation. However at the moment the car must not lose sight of the line of sight, otherwise it will not

be able to steer anymore. That's why we would have liked to use the GPS implementation we could have done. Indeed our idea would have been to use the previous GPS positions that would have been made in previous laps in order to be able to come back on the line in case we lost it. This would have meant that at every moment the car would have had a command to give to the actuators.

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.

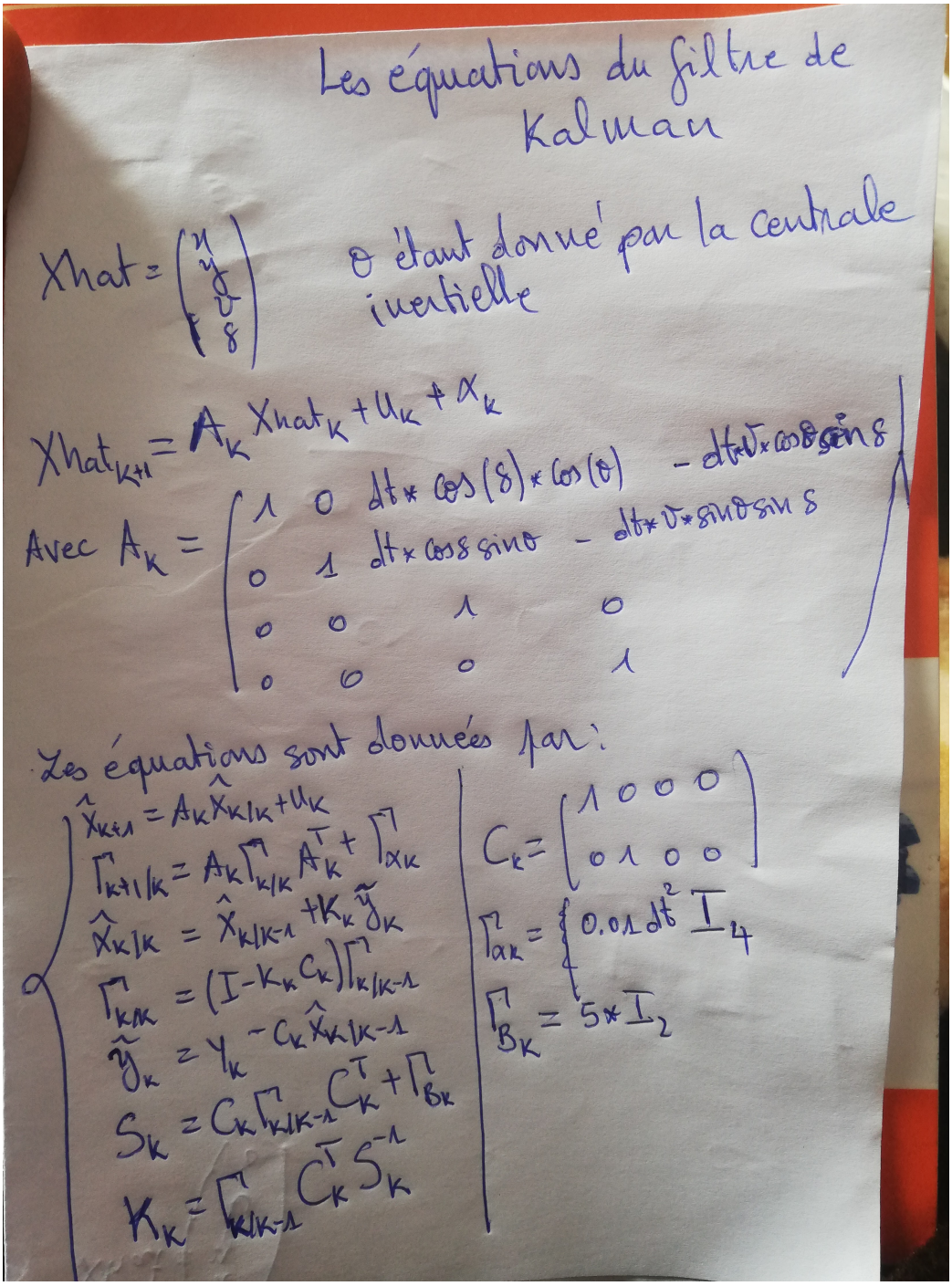


Figure 12: Kalman Filter Equation

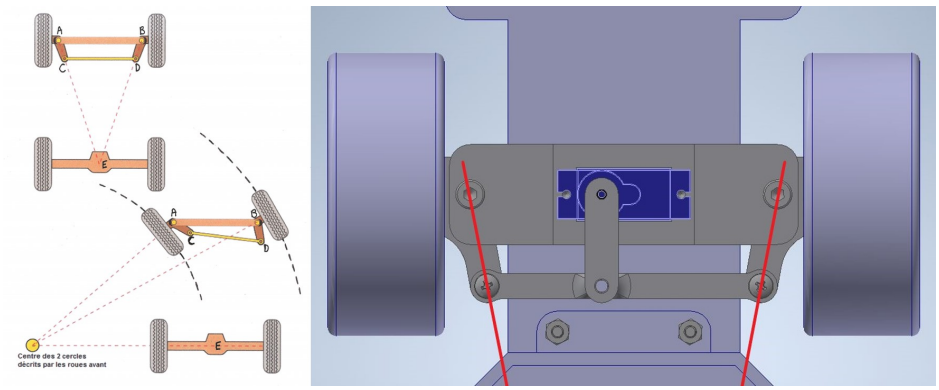


Figure 13: Steering system

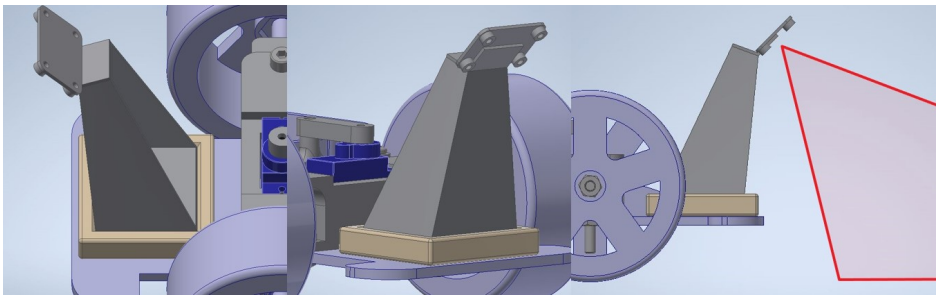


Figure 14: Front camera support structure

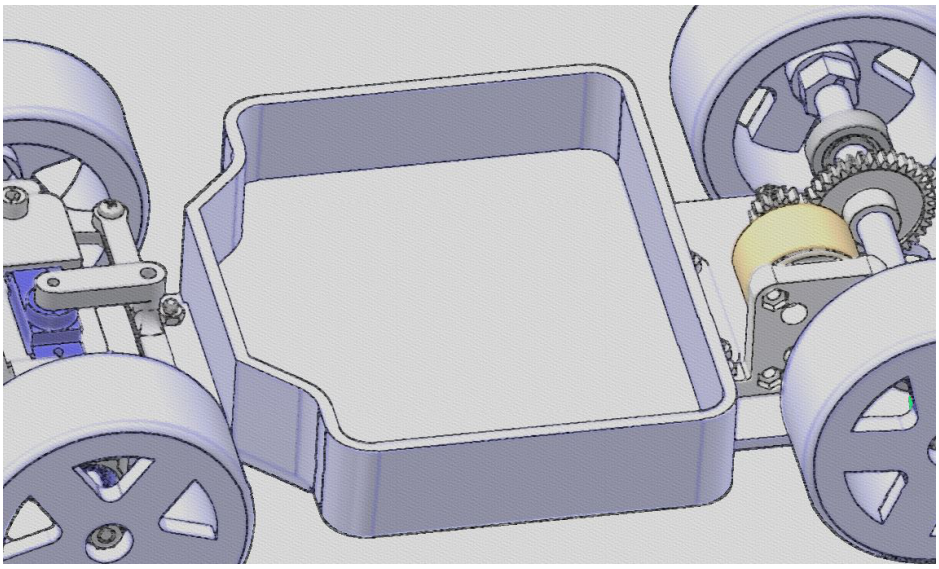


Figure 15: Central component box

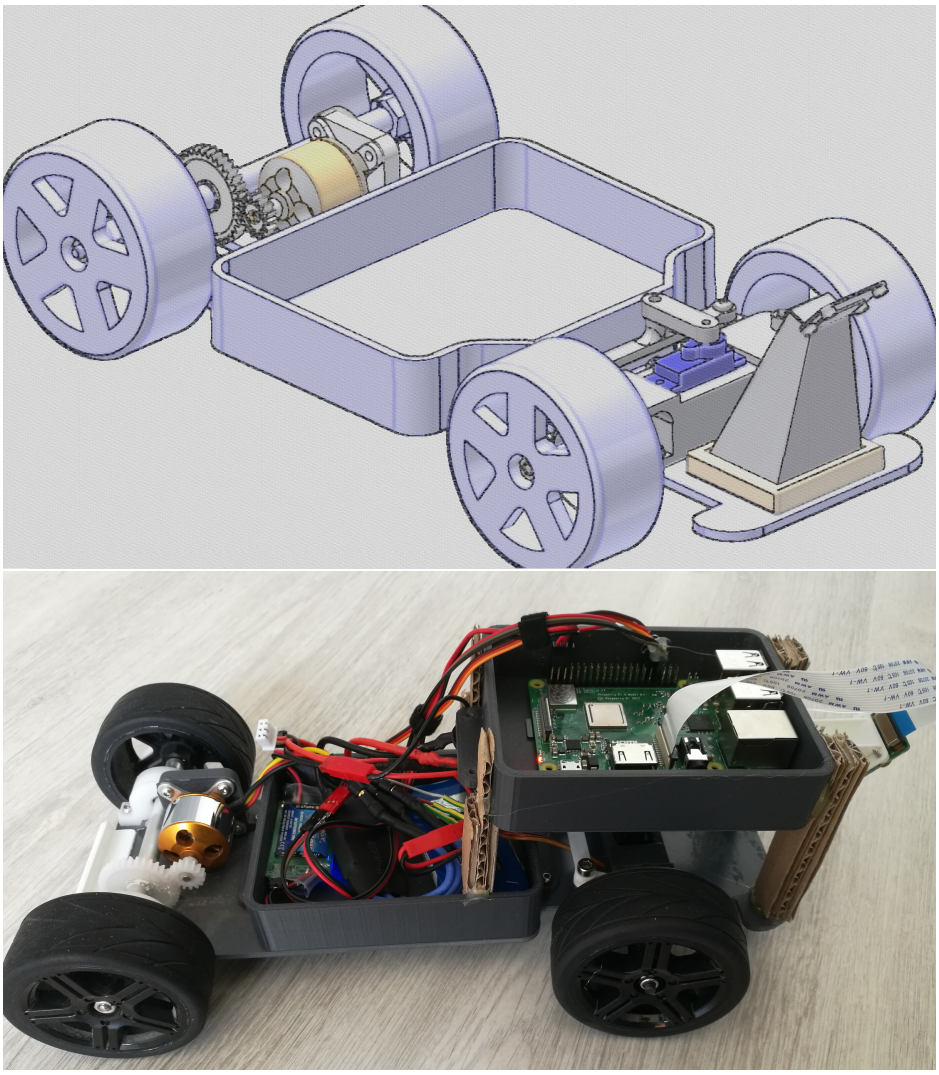


Figure 16: Expextations and reality