

Active surveillance

INTERNSHIP REPORT

Liviu VALACHE, IUT of Cachan, GEII ESE
Tutor : M. Emmanuel LOMBARD
Reference teacher: M. Julien GABIOT

université
PARIS-SACLAY



IUT DE CACHAN

Thankings

I want to express my gratitude to the people I worked and exchanged with during this internship,

To Mr. Julien Gabiot for the initial post offer, material supply and his general inputs, ideas and solutions.

To Ms. Alavi Roxane for accepting me to this post and for her support regarding the future of the project.

Emmanuel LOMBARD the maintenance workshop site manager of the and our internship tutor for his operativity on site on our demands about documents, materials and information.

I value our exchange with the Guards for their input on this project as well as offering us information and answers to various related questions.

The QSE and Operator teams for their daily politeness and enthusiasm towards our project.

And I also want to thank my comrades Marème KANE and Vuk MILIC for their efforts and passion to their objectives in this project, especially when completing common goals or for informational exchanges.

Internship introduction

To complete our second year at the IUT we must partake in an internship anywhere, but ideally to an external entity to our university. The mere work at said internship should prove the competences we have acquired up to this point: company research, communication, preparations and finally the practical experience. Furthermore, as a student of embedded systems and electronics, our internship work has to correspond to our domain.

In my case the social part of the process wasn't that smooth, and as a bailout option I was able to pass this internship at the Parisian Autonomous Transport Administration – RATP group thanks to Mr. Gabiot and Ms. Roxane.

The foundation of the project for this internship option was the question of the development of a more capable security solution, and its experimentation at the Choisy workshop of the RATP group. Together with the collaboration of the Cachan IUT the project was defined and launched, allowing students to commence its development.

During the ten-week internship we will have researched and learnt the environment, requirements and developed tools for its further continuation. This being a long and fresh project, our team had to define the technical solutions to answer the original requirements. In the process, we had to organize our individual tasks and work together to achieve multiple checkpoints during this period.

Each one of us three had initial directions for their work starting from the main ideas that expanded and defined themselves as they were pursued. My two [missions](#) that involved software development and networking have grown over the course of the internship and are now serving the base for the [future](#) work that this project defines.

This is a general project report, no code, logic structures or experimental features are presented here.

Contents

The company and mission	5
RATP group overview	5
Concerned sector	6
Choisy workshop	6
The mission	7
Organization	8
Mobile surveillance task	9
Robot platform	9
Onboard electronics	10
Software	11
Lidar	12
Ros bridge	12
Slam toolbox	12
Navigation 2	13
Launch files	13
Network and server task	14
Physical conditions	15
The LAN setup	16
Video processing:	17
Video storage:	17
The server:	18
Frigate:	18
Home Assistant:	19
Human-Machine Interface:	19
Conclusion:	20

The company and mission

RATP group overview

The Parisian Autonomous Transport Administration (RATP) is one of the largest public transport operators in the world, fully owned by the French State. It was founded in 1949 to consolidate the Paris transport routes that were active after WW2. Based in Paris, the group operates an extensive network across Île-de-France, including the entire Paris Métro (16 lines), part of the RER network, 9 tram lines, and over 350 bus routes.

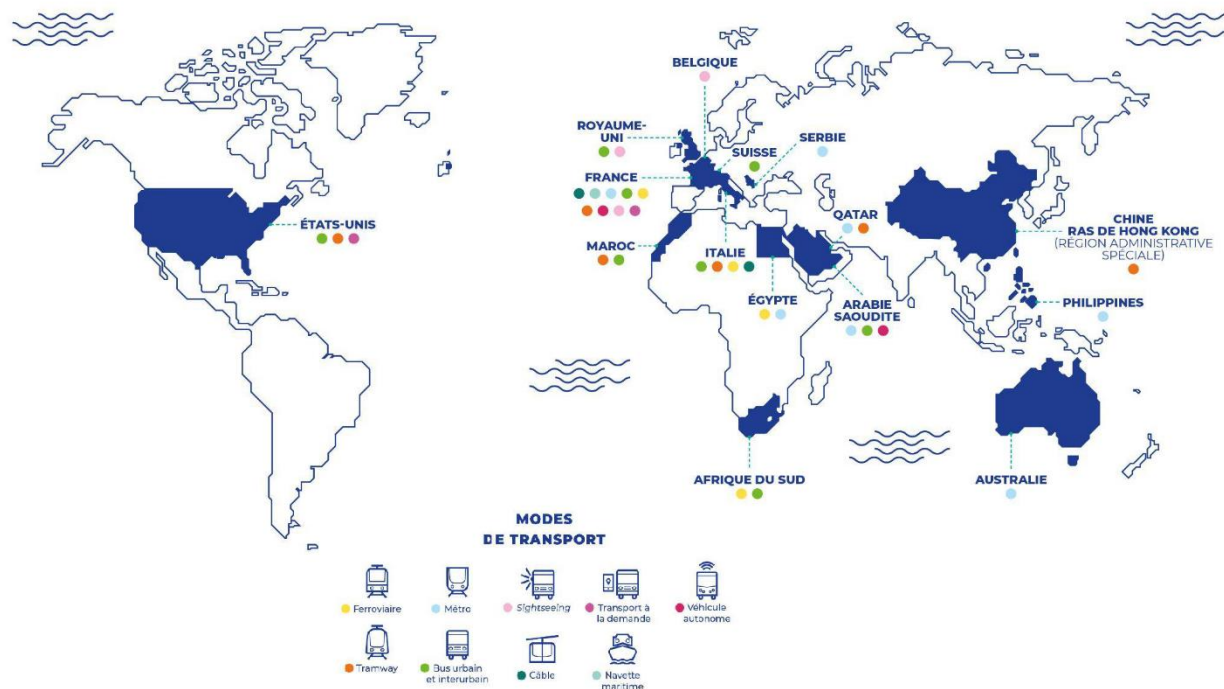


Figure 1 International presence

Together, these services support around 12 million passenger journeys per day. Beyond Paris, through its subsidiary RATP Dev, the group is active in 14 countries across Europe, Africa, the Middle East, Asia, and North America, managing modes that range from buses and metros to trams and even cable cars. With a workforce of about 70,000 people and annual revenues around €6–7 billion reinforcing its role as a leader in urban mobility.

Concerned sector

Within the RATP, our internship workshop falls in the Département Matériel Roulant Ferroviaire (MRF) department, it is a specialized division responsible for the lifecycle management of all rail rolling stock, including the Paris Métro's MF and MP train series, the RER operated by RATP, and the tramway fleet. They handle preventive and corrective maintenance to heavy overhauls, mid-life modernizations, and the commissioning of new trains such as the upcoming MF 19. The department also manages the end-of-life process, ensuring dismantling and recycling of retired stock, like the MF 67 units processed at Choisy.

MRF coordinates several workshops across the Paris region, each with defined expertise, creating a distributed but complementary network. With responsibility for more than 5894 rail vehicles managed by about 3290 workers, it plays a central role in ensuring daily reliability for millions of passengers.

Choisy workshop

The Choisy workshop is one of RATP's major maintenance facilities, positioned within the Département Matériel Roulant Ferroviaire as previously stated. Its primary responsibility is the full maintenance of the MF 77 fleet operating on Line 7 with its train maintenance workshop (AMT), one of the busiest lines of the Paris network.



Figure 2 AMT, endline maintenance



Figure 3 AMP, heavy maintenance

In addition, Choisy's patrimonial maintenance workshop (AMP) performs major overhauls on other models such as the MF 67, MF 88, and MF 01, and it also manages tramway bogie revisions for some lines. The site is further tasked with dismantling retired MF 67 units, reflecting its role across the full lifecycle of rolling stock. Choisy combines heavy maintenance, revision, and deconstruction capabilities, making it a strategic asset for ensuring fleet reliability, extending vehicle lifespans, and supporting network modernization. Its integrated functions place it at the crossroads between daily operational continuity and the long-term renewal of Paris's metro and tram fleets.

The mission

At the moment of the internship, the Choisy workshop site was suffering from the illegal intrusion of people that would cause material damage periodically.

With 341 300 passengers per day on the line M7 and up to 500K€ per day of work delay, these disruptions are dangerous and unacceptable.

The situation is worsened by the presence of at least 5 [vulnerable zones](#) in relation to the geographical situation: blind spots, public zones, buildings as part of the perimeter fence.

Both physical obstacles and surveillance systems have been put in place as a consequence, but over time the security equipment began to degrade: legacy software, failing disks, cables and possibly cameras have fallen out of order.

The current security system bills 80K \$ to maintain, and our task is to not only come up with a cheaper solution but to also improve it using autonomous robots, AI detection and warnings to emphasize active surveillance.

The idea in the beginning is to have both stationary wall mounted cameras and mobile ones on autonomous robots connect to a local server for active human intruders detection and video storage.

The intruder notification warning system can be anything from phone calls, PC/ phone messages, emails or on-site alarms.

This setup would alert any intrusion and possibly prevent future material damage compared to the previous system where the infraction could only be recorder and reviewed afterwards.

From the beginning my mission was the software part of the robot's automation and network hardware setup.

Behind these tasks sit mainly research, material selection and latter development:

The local infrastructure has to be analyzed, solutions proposed and materials selected for further steps. The work on this task is recorded in the respective [chapter](#) from page 14.

Suitable robots have to be selected with respect to terrain, autonomy, hardware and software support. The development environment has to be studied and used for building the programs enabling the autonomous robots' operation, the task is described [here](#) from page 11.

Organization

During the period of the internship, because of the lack of materials due to delayed deliveries, the project physically cannot be finished. This is the reason why for the best efficiency it's reasonable to view this internship as preparation for the total project.

In the Figure 3, I propose the general plan for this project starting with the internship and ending at the end of August.

Milestones mark the end of a fundamental step that is normally checked with a status report (the **deep-RED** cases).

The **faint-RED** cases under each month depict the weeks that we are present at the University, respective to our work-study planning of the year.

In this plan, we're supposed to receive and have confirmed working all the required materials by the end of **October**;

Complete secondary functions like network setup, camera installations, the robot backpack, and various software related ones by the end of **March**;

Finalize the integration of the primary function of the project (working state) by the end of **July**;

For the last month of **August** our task is to monitor, patch and optimize the full setup as well as produce instructions for the maintenance and professional use.

Zoom in for a bigger picture.

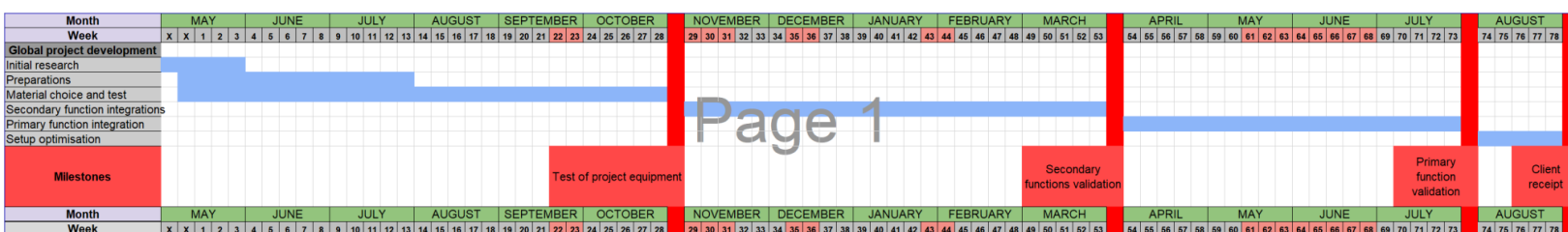


Figure 4 Project schedule




Mobile surveillance task

Since the [territory to survey](#) is large and camera placements are someplace unreasonable, an idea is to mount the cameras on some autonomous robots that will move around on a pre-defined path. The dog-like robots are picked over wheel-based ones as the territory isn't all flat and the dog robots can walk over debris or small obstacles.

All robots will use the same software, just on different territories. They will follow set paths with IP cameras and various sensors onboard. Ideally, they are all connected to the same local network and communicate with the main server, but on too large terrains where normal WIFI coverage isn't possible we'll implement exotic solutions.

Robot platform

The robots we will be using are:

		
Unitree GO1	Unitree GO2	Deeprobotics Lite3

Initially we have the Unitree GO1, but we need more robots to cover all [risk zones](#). This is why after researching other robot options (GO1 discontinued), we chose the Deeprobotics Lite3 for its hardware and software support. Yet, the Unitree GO2 which is cheaper remains tempting for its unofficial but large community support. ~~Extensive analysis in the 'prjData' excel file.~~

Onboard electronics

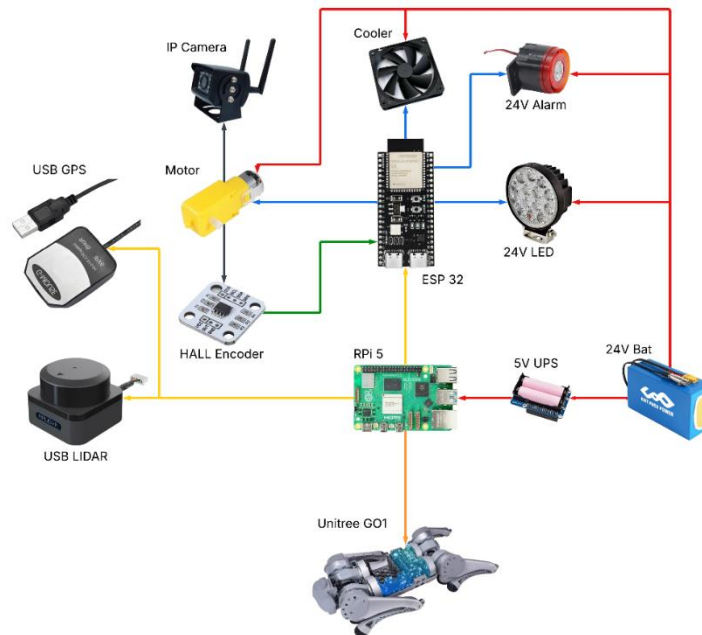


Figure 5 Onboard electronics

The robot by itself in the context of the mission is nothing but a collection of motors with some included electronics to drive them. In order to achieve our goals, we need to add a ‘brain’ – the Raspberry Pi 5 computer. The main navigation functions and other logic runs directly on this computer. While we mentioned the robot as the assembly of motors – essentially the ‘muscles’ of the platform, it needs ‘eyes’ in order to navigate in the environment. These are fulfilled mainly by the LiDAR, a 360° laser scanner, and partly by the camera, that not only doubles as a security video feed but also helps the robot localize itself on the territory.

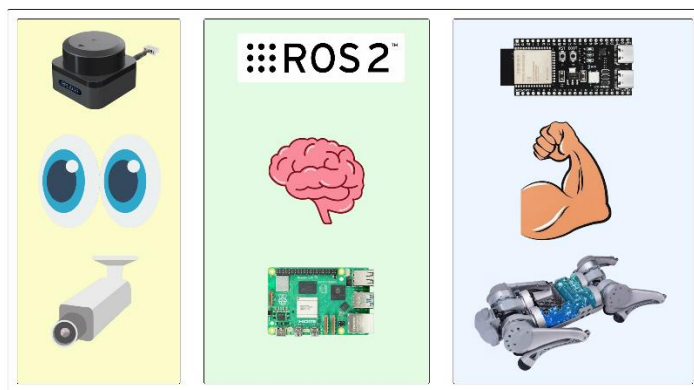


Figure 6 Robot general analogy

In the [onboard electronics figure](#), we have the previously mentioned devices: robot, computer, LiDAR and camera, but there are more present.

The power supply is easy to understand, the 24V battery provides power to the installed electronics and the UPS allows the computer to operate individually for redundancy and ease of development.

The additional electronics though are modular, and are installed based on the user's requirements and the robot's operation zone. This is possible by the addition of an ESP32 microcontroller that interfaces and controls these devices.

Currently the diagram represents a basic setup with a theft resistance and alarm module:

If the robot detects an intruder, senses physical manipulation or is taken outside its operating zone, it will alert the local server and engage its LED and alarm combo to deter the intruder and alert the staff. In addition, if it suffers lethal damage the server can also detect this and engage the alarm system.

Future models could be equipped with more sensors like IR, radar or audio as well as more customized actuators like IR illuminators, voice speakers, projectile launchers and pyrotechnic devices.

Software

The software side of the autonomous robots gives us freedom for development, yet even if writing specific scripts for our tasks might be easier to work through, scaling up we'd need a more universal solution.

This is where ROS 2 (Robot Operating System) comes into play, without reinventing the wheel it allows us to render these robots autonomous for our tasks with all the additional hardware of our choice.

At the time a simple installation for ROS jazzy wasn't present, it had to be installed from source as described:

<https://docs.ros.org/en/jazzy/Installation/Alternatives/Ubuntu-Development-Setup.html>

The basics of how ROS works:

A project is usually stored under the form of a ROS package: in this package there can be any types of files needed for the robot's function. Usually packages encompass configurations, launchers, definitions and scripts for operation modes and others. These scripts often called nodes and only certain required ones are activated during the operation of the robot, it is common for packages to contain numerous nodes.

Big assets like slam toolbox, or navigation 2, premade configurations or the programs to run specific hardware use their own packages to remain organized and modular for development.

Given this, it's not uncommon for robots to have multiple packages installed, the logistical issue this poses is that each node has to be called separately from each needed package manually.

Lidar

We have a fair lidar available to use, the RPLidar S3 with a max range of 40 meters, 10 Hz data output and decent point density.

For this Lidar we can use the official ROS2 package, though it has to be built from source since our ROS 2 version is newer.

In this package we find the node for the respective Lidar to use.

Ros bridge

The dog model can be interfaced with using either ROS 1, MQTT (limited) or UDP messages. Logically we want to use the existing ROS infrastructure for the communication.

Since both ROS revisions use different protocols for communication a package for bridging the two is necessary. The one used here is specific for this robot and not official, yet it exposes the same ROS topics as the robot.

Using this bridge, we can send movement, LED and audio commands all corresponding to the ROS 2 standard, in return we get the data of each of the robot's sensors like ultrasonic, position encoders, imu, cameras, battery state and possibly microphone.

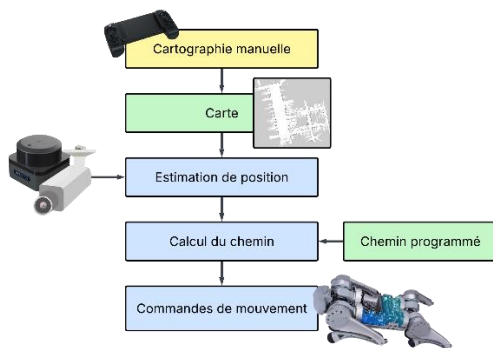
Slam toolbox

There are multiple ways to create an environment map, hector slam which works best for Lidar-only robots, but it's limited to ROS 1. There are homemade mapping options but none for our ROS version. The last option remains slam toolbox, which can be built from source for our version. With some configurations it works with our materials.

Navigation 2

Nav 2 is another fantastic package including dozens of nodes for any desired behavior, it allows the robot to load a saved map, localize itself and navigate on the territory. This package is built officially for our ROS2 version and thus the installation is default.

For it to work, it requires a pre-made map of the territory and a planned path to follow. The rest is handled by it, like the obstacle avoidance, position estimation and checkpoint following.



In **yellow** is the only external manual operation required, but it could also be automated.

In **green** are the files/ configurations we define for the robot's territory.

And in **blue** is the logic loop that Navigation 2 is repeating to achieve autonomous navigation.

Launch files

For a complete navigation stack launch, there are about 7 nodes that need to be activated, including the map server, adaptive monte carlo localization, costmap generators, navigation server and so on. In consequence, to launch a stack about 8 SSH sessions with terminals are required for each test.

To ease development and latter deployment, three dedicated ROS2 launch files were created inside a standard main package to streamline the process:

- `mapping_launch.py` brings up the LiDAR and SLAM nodes to create or refine maps of the environment.
- `localization_bringup_launch.py` starts AMCL together with the map server, enabling the robot to localize itself on a previously saved map.
- `navigation_bringup_launch.py` combines localization with the full Nav2 navigation stack, including costmaps, planners, controllers, and lifecycle management, allowing the robot to autonomously move to goal poses.

By consolidating multiple nodes and parameters into single commands, these launchers significantly reduce setup complexity (replacing several terminal windows) and provide a repeatable, maintainable way to start the robot in different operational modes (mapping, localization, navigation).

Network and server task

The high-level idea concerning this task is to have an independent network (separate from RATP) to which the local security devices would connect. The same storage behavior is expected as before, and in addition a human detection system should be working during the closed hours. Robots with sensors are also expected, together with a total setup overview interface and alarm mechanism.

Upon arriving to the project, my teammates were already working on it. On this topic, only a detection demo and the minimum connectivity for it was present.

I needed to confirm the viability of the following points:

- Overall connectivity of all devices.
- The server receives video streams from all the cameras, process and store them.
- The server houses the main HMI of the entire system, including the robots.

The HMI will be offered in the form of a website that should give the user information about the security situation, robots statuses and their position on a map. Possible commands involve alarm response control, robot navigation as well as high level mode commands (return to base, change position, etc).

My work here encompasses the analysis of the initial idea and the present devices on the site it concerns. Then the creation of a [network setup](#), material logistics and their installation.

Here I describe the choice of materials for the [network devices](#), the [server](#) and the [device configurations](#) and [software structure](#).

Physical conditions

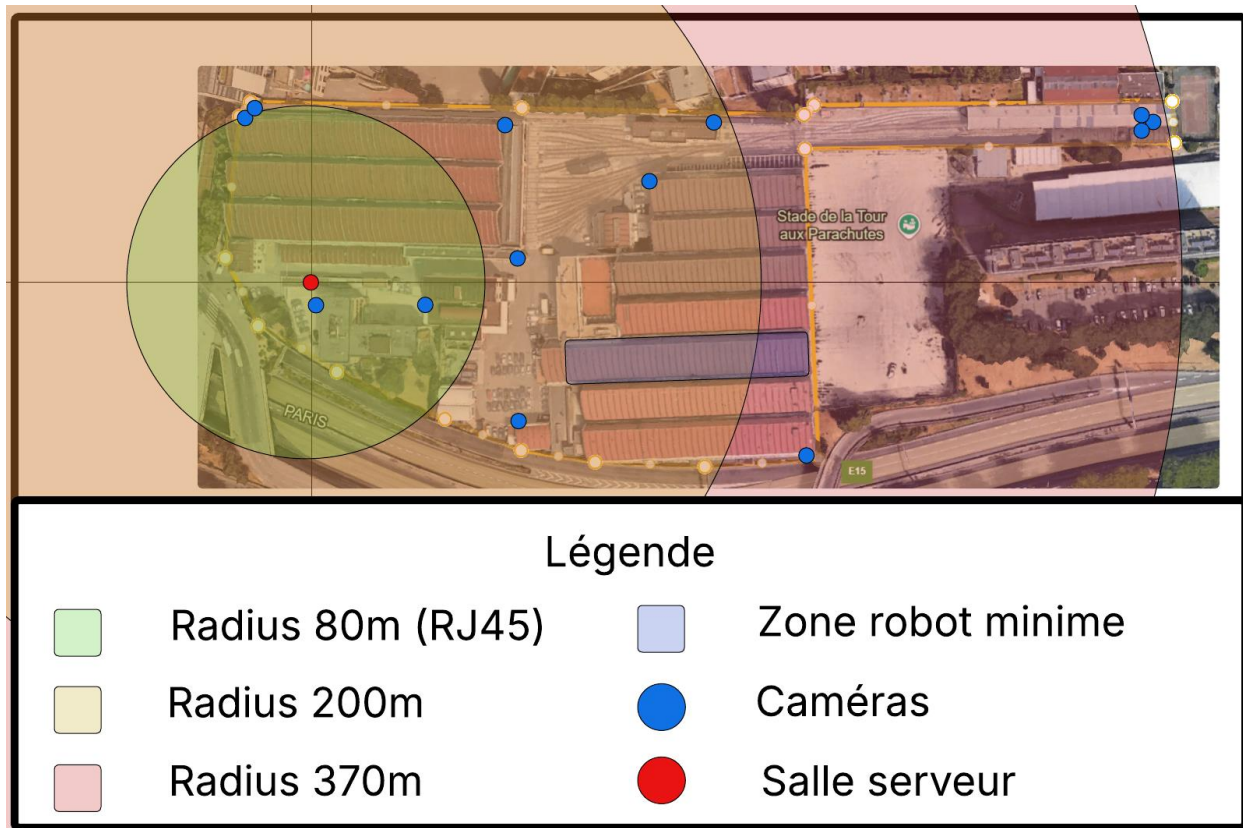


Figure 7 LAN device locations map

On the proposed map we get a first look on the current placement of the majority of our cameras as well as the first robot's operating zone.

We have cameras spread out as far as 370 meters away from the server and a large zone for the robot that will have to have wireless coverage.

The majority of the devices are out of range of the traditional RJ45 ethernet cable, we are forced to use either optic fiber or make use of the pre-existing coaxial cables.

The LAN setup

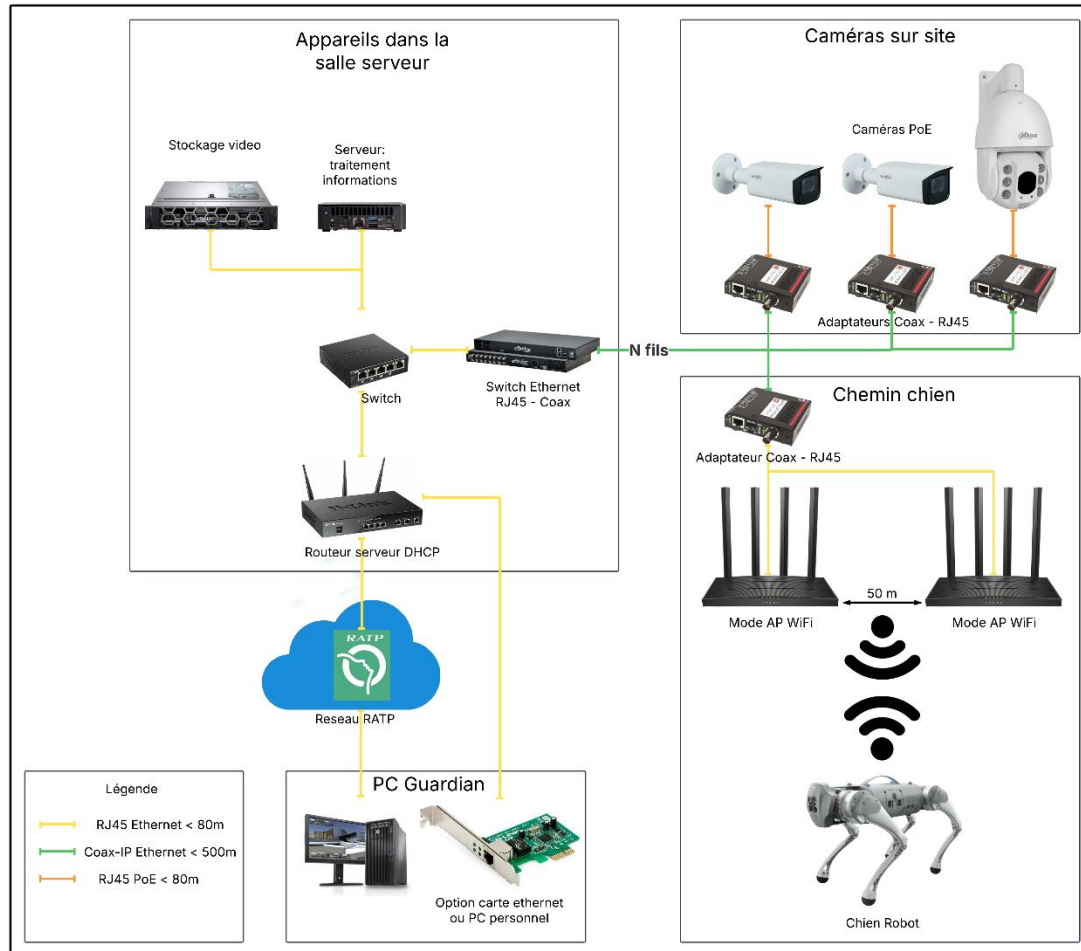


Figure 8 Network setup diagram

This diagram presents the setup of our LAN network, where we use the coaxial cables to reach devices in remote locations.

Starting with the DHCP server, it provides IP addresses (ideally static) to each connected device. The server combo here takes care of the storage, processing and HMI.

The guardian is able to control and assess the state of the system from the personal PC using the HMI and view the camera streams at the same time.

Remote devices are linked to the DHCP server through switches and adaptors using coaxial and RJ45 cables.

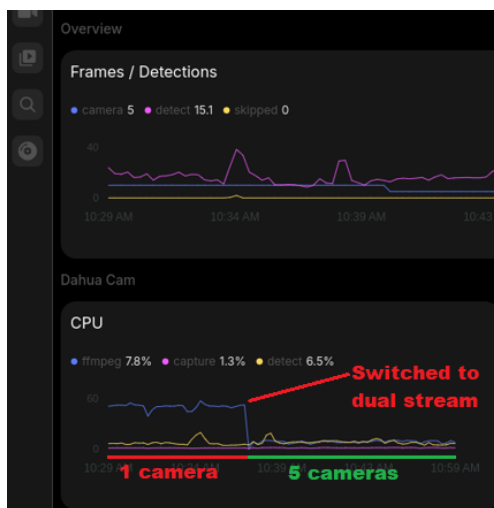
For the robot's wireless zone, we use a mesh of AP mode routers to provide uninterrupted access to LAN. Concerning the security of the wireless setup, we will use WPA2-Enterprise certificate identification and only the 5 GHz band.

Video processing:

The USB Google Coral at it's maximum theoretical performance can handle **150 FPS** detection on **300x300 frames**.

The first issue is that the Coral can essentially handle about **7.5** (= 150 FPS / 20 FPS) streams from our cameras. This can be improved at the cost of degrading the stream framerates.

Secondly, the Coral only processes 300x300 pixel frames, which means the PC's CPU has to downsize each frame beforehand resulting in high load:



~ 57% CPU running with a single stream.

The solution is to enable dual streams on the cameras with the original HD 720p20 stream for storage and a SD 360p5 stream for detection. This makes the cameras handle their own downsizing freeing up the PC's CPU:

~ 10% CPU treating 5 video streams at the same time.

Video storage:

On the territory there will be about 15 – 17 wall mounted cameras, with more from each robot dog. Keeping with the previous surveillance setup's characteristics the videos will be stored for 15-day periods, in HD 20 FPS.

One stream detail:

Resolution	HD (1280x720)
Framerate	20 FPS
Codec	H.264
Bitrate	2Mbps ~ 0.25 MBps
Storage per day	21.6 GB = 0.25 MBps * 86400 seconds/ day
15-day storage	324 GB = 21.6 GB * 15 days

With an estimate of 20 cameras online we would need around **6.5 TB of disk space**.

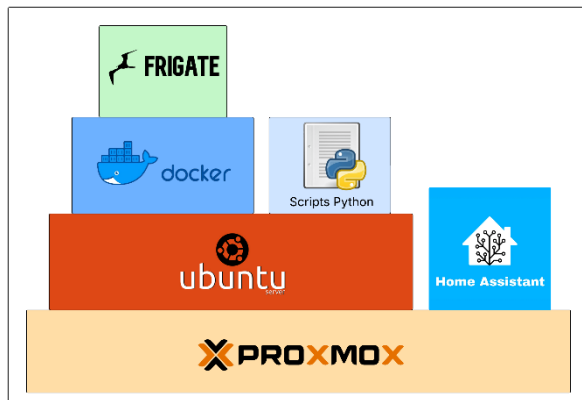
For some minimal redundancy we could opt for **two 8 TB HDDs**.

The server:

After our optimization, the PC inhibiting Frigate doesn't need to be too powerful, the 4 core CPU we have should suffice. However, the storage HDDs cannot be installed on it, and an external machine becomes a necessity now.

At this point we can look towards a more practical rack-type server that with Proxmox running, could take care of both localizing the storage drives and allow us to fine tune resource allocation to the various tasks (Frigate, Home assistant, HMI, storage) in a centralized manner.

Otherwise a second PC with storage means is needed, and new tests for bandwidths, CPU utilizations and setup are required for validation.



We can imagine a Virtual Machine with 4 cores and about 8 GB Ram would suffice the Frigate service, robot's service and HMI, another weaker for the Home assistant.

Any 8+ core Intel Xeon, 16 GB Ram, $\geq 2 \times 3.5''$ caddies for HDDs with USB 3 ports for the Google Coral server would fit these tasks.

Figure 9 proposed server structure

Frigate:

Frigate is a NVR container-based application that runs primarily on Docker, meaning it can effortlessly be deployed on either Windows or Linux based operating systems.

In our setup, all the video streams on the network pass through Frigate, and it handles the storage and detection on separate streams.

Once installed in a container, the simple way to set up a basic working state is to complete a yaml configuration file to specify parameters like video sources, resolutions + fps, storage, detection devices and others.

While only CPU-based detection is possible, it requires serious resource dedication and high power use. Instead an edge TPU is recommended to handle the detection tasks way more efficiently, these can be either Google Coral, Hailo or other devices that come in either USB or M.2 connection variants.

Home Assistant:

Home assistant is a powerful automation software with an enormous amount of integrations and device compatibility. It allows for basic automation through it's graphic interface as well as heavy scripting using yaml configuration files.

For us it is used in a simple way to catch the detection mqtt messages from the Frigate service and send an alert email.

Further it will be used to rotate the PTZ (360°) cameras to enhance their effectiveness using yaml-scripted HTML commands.

It can also be potentially used as a simple user interface for a limited part of the system, but we will be using a dedicated web-page for this.

Human-Machine Interface:

There are going to be multiple web-page based interfaces: the main server-based one that will contain all possible information and controls, and a local limited interface for each robot.

While the current interface is not in a working state yet, the possibilities and required functionalities have been researched and defined.

All the mentioned features will be present on the main webpage HMI, but for the robot-limited HMI only the features in **green**:

- Live visual map showing robot's positions with ROS2 integration
- Live GPS-es positions on the asymmetric territory with theft logic
- Robot's states (battery, mode, temperatures, sensors)
- Robot commands (movement, checkpoint, modes)
- Complete system detection and alarm status
- Server state (disks, CPU, temperature)

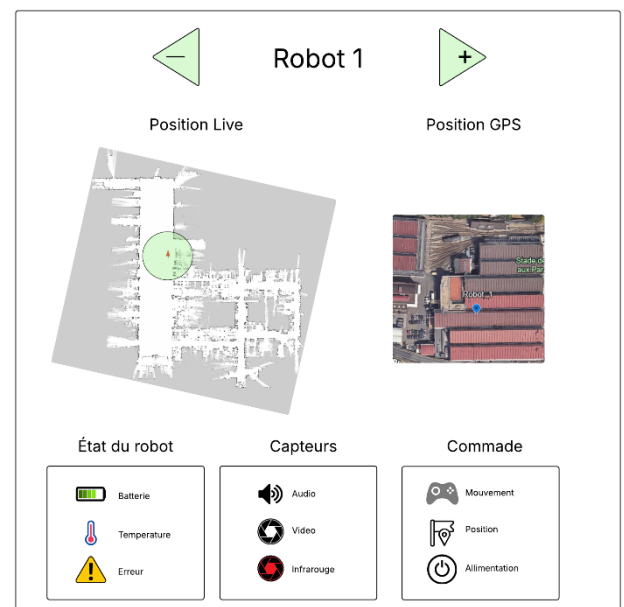


Figure 10 Concept Web UI

Conclusion:

This is the end of my internship at the RATP Patrimonial Maintenance Workshop. Here I had the pleasure of working on an innovative and needed project for the security of the site. It was an unexpected but welcome research and development mission, where we touched many of the usually studied topics at the university, more mechantronical and higher level than what we're used to in truth. During my work I particularly liked discovering and mastering the ROS environment that massively simplifies large scale systems with many components. Understanding that this is a preparation phase in the scale of the project, I pivoted on laying a standard foundation and the tools necessary for further development.

In the context of the company it was interesting to see the dynamics for taking decisions, like virtual and personal meetings for informing and exchanging ideas, material reports and presentations for validations and demonstrations.

Specifically, in such an autonomous setting, the need of proper long-term organization and deep research have highlighted themselves to me.

As for the prospects of this project, the research is all complete and needed materials defined. Upon continuing this project again, we are going to acquire these materials and devices, run tests and install them as required.



Figure 11 Future objectives

Figure table

Figure 1 International presence.....	5
Figure 2 AMT, endline maintenance.....	6
Figure 3 AMP, heavy maintenance	6
Figure 4 Project schedule.....	8
Figure 5 Onboard electronics.....	10
Figure 6 Robot general analogy	10
Figure 7 LAN device locations map.....	15
Figure 8 Network setup diagram	16
Figure 9 proposed server structure	18
Figure 10 Concept Web UI	19
Figure 11 Future objectives	20
Figure 12 Annex 1: intrusion zones.....	22
Figure 13 Annex 2: ORGANIZATION CHART of the RATP group.....	22
Figure 14 Annex 3: ORGANIZATION CHART of the MRF department.....	23
Figure 15 Annex 4: ORGANIZATION CHART of the MF67-TW department	23
Figure 16 Annex 5: Project Gantt.....	24

Annexes



Figure 12 Annex 1: intrusion zones

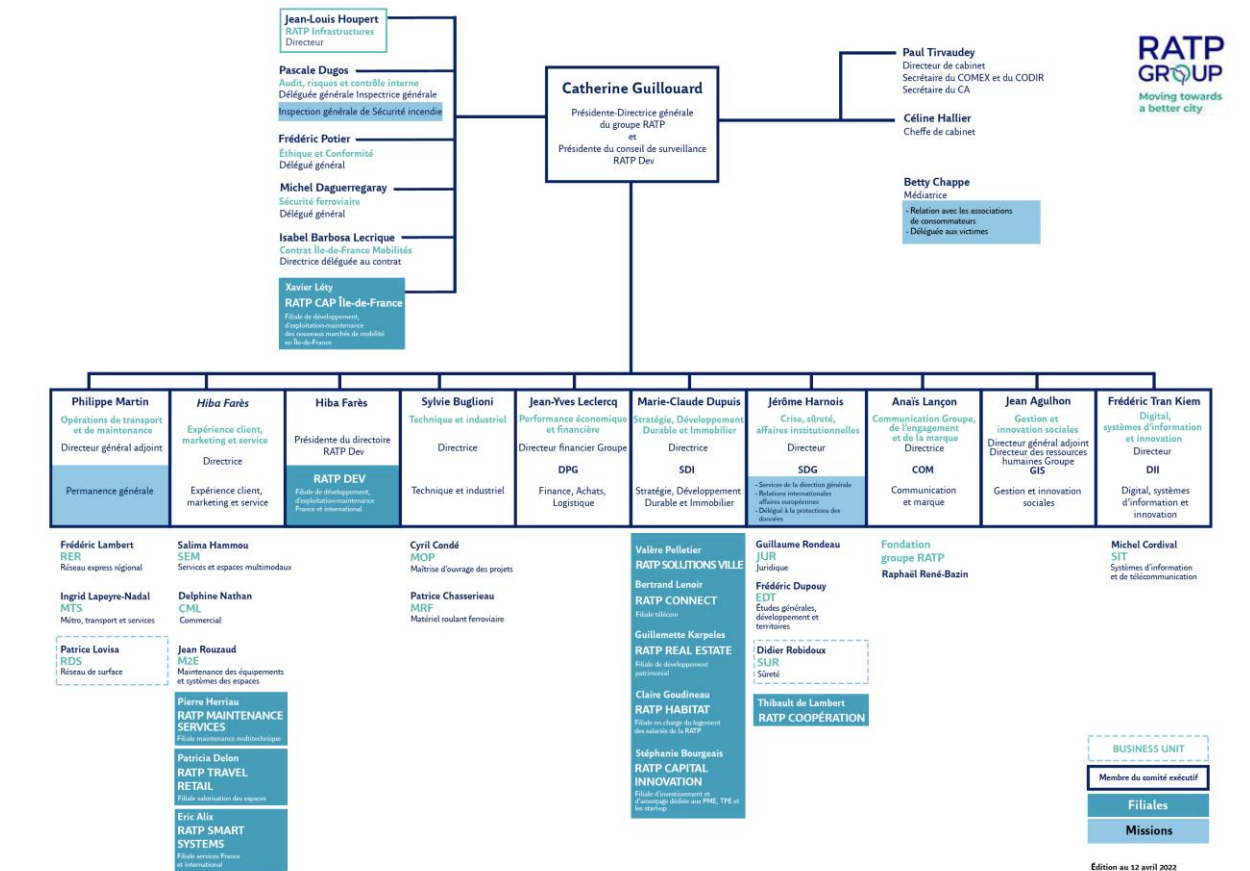


Figure 13 Annex 2: ORGANIZATION CHART of the RATP group

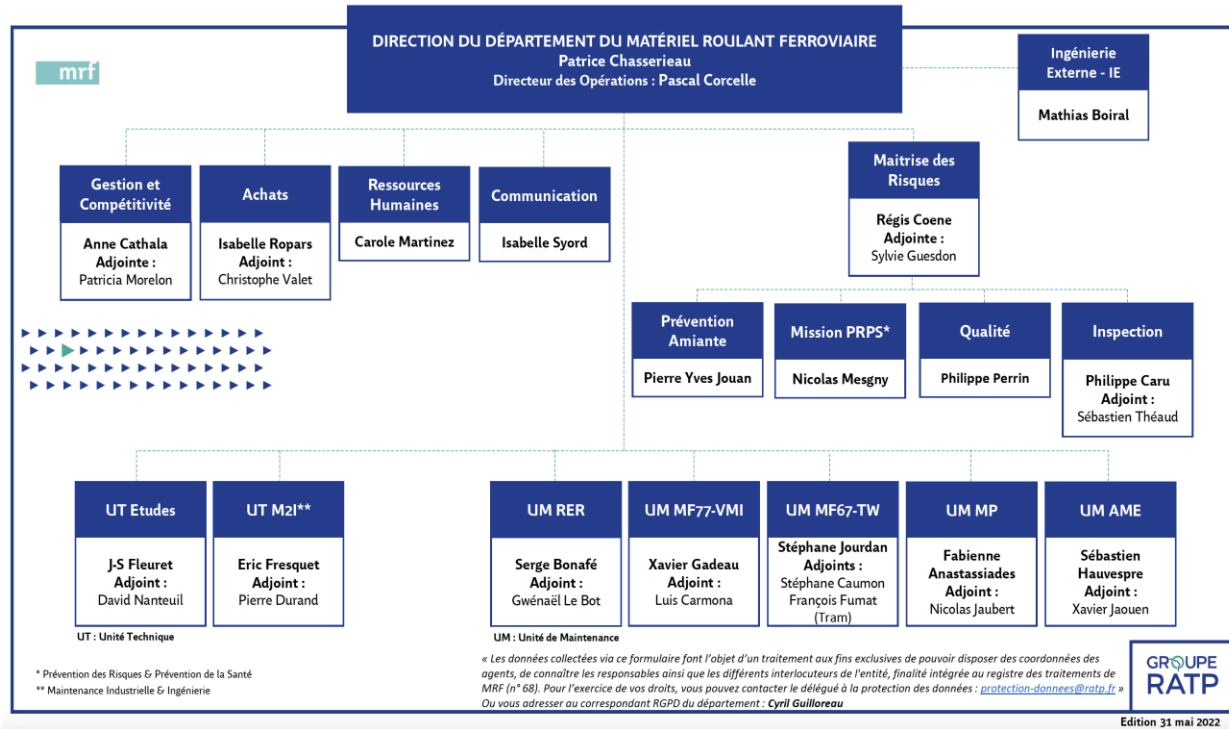


Figure 14 Annex 3: ORGANIZATION CHART of the MRF department

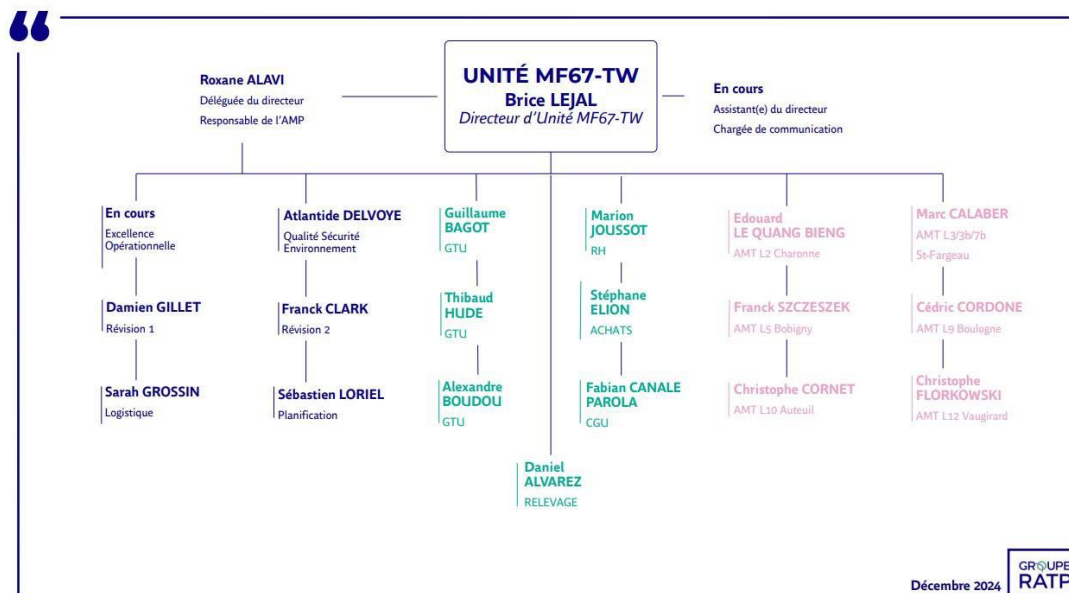


Figure 15 Annex 4: ORGANIZATION CHART of the MF67-TW department

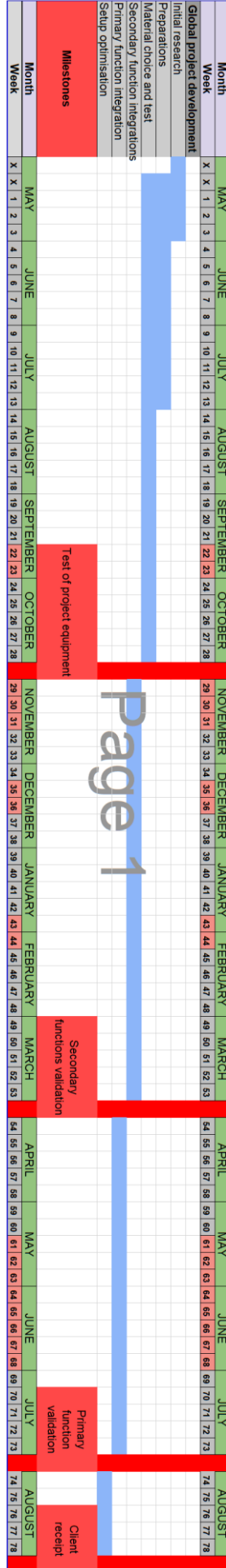


Figure 16 Annex 5: Project Gantt

Frequent sources:

Backdoors GO1: <https://www.youtube.com/watch?v=IAuvVsZrVMA>

nav2:

https://docs.nav2.org/getting_started/index.html

<https://docs.nav2.org/plugins/index.html>

https://docs.nav2.org/setup_guides/sensors/mapping_localization.html

https://docs.nav2.org/tutorials/docs/navigation2_with_slam.html

costmap: https://docs.nav2.org/plugin_tutorials/docs/writing_new_costmap2d_plugin.html

amcl: <https://docs.nav2.org/configuration/packages/configuring-amcl.html>

ros2 inst: <https://docs.ros.org/en/jazzy/Installation/Alternatives/Ubuntu-Development-Setup.html>

gps: https://docs.ros.org/en/api/robot_localization/html/integrating_gps.html

slambox: https://github.com/SteveMacenski/slam_toolbox?tab=readme-ov-file

Ros Go1: https://docs.trossenrobotics.com/unitree_go1_docs/getting_started/ros.html

slidar:

https://github.com/Slamtec/slidar_ros2

https://github.com/slamtec/rplidar_ros