Pave Guard slides presentation

[formal greetings]

First part

Road maintenance as of now…

[FIL]

Now, road maintenance has two main problems that we must handle:

Management: It is not known in advance which roads will require maintenance in the near future. Typically, repairs are carried out only when the damage is severe, leading to higher costs and more extensive interventions.

Planning: After a major storm, it's not just a single prejudiced road that requires repaving, but many. As a result, maintenance takes a long time to complete—potentially leaving roads vulnerable when the next storm arrives.

Delays: Roads typically remain in poor condition for three to six months, leading to safety risks and vehicle damage. Additionally, prolonged maintenance periods result in dissatisfaction and inconvenience.

Difficulties in management and planning cause that 20% of interventions are *emergency interventions*, which have 190% extra cost.

Our solution

[FIL]

We offer PaveGuard, which is a completely automatized, plug-and-play, cost-efficient, road maintenance planning system.

It continuously monitors road conditions, detects early signs of damage, and optimizes maintenance scheduling, reducing repair costs and delays.

By leveraging smart sensors and predictive analytics, PaveGuard helps to prioritize interventions efficiently, ensuring safer roads and minimizing unnecessary expenses.

Advantages of PaveGuard

[FIL]

The advantages of PaveGuard are, first of all, very low production and installation costs.

In addiction, we must consider that PaveGuard system performs a fully-automated track of road conditions, reducing number of needed humans supervisors.

Therefore, adopting PaveGuard we can save a lot of moneys also reducing emergency maintenances thanks to preventive maintenance.

Obviously, more maintenances imply safer roads, decreasing road accident risks.

Process timeline

[NIC]

Here, you can see an overview of the entire process.

First of all, data are collected from environment using a set of devices, which have special sensors to capture roads conditions.

Then, data are elaborated by our system. In particular, we have trained a specific AI model that is able to predict future state of the roads.

Thanks to predictions, we are able to provide useful information to make more conscious planning of maintenance interventions.

In the end, directly from our dashboard, interventions can be planned and nextly they can be executed.

Resources

[NIC]

To make that possible, PaveGuard is composed by several components.

StaticGuards are fixed monitoring devices designed to continuously measure atmospheric conditions and traffic volume.

DynamicGuards are deployed monitoring devices able to capture the road status for assessing the pavement quality. They are installed directly on citizens vehicles.

Dashboard allows managers to analyze future road status and to plan maintenances beforehand.

Future plans

[NIC]

Our main four future plans for PaveGuard are:

Introduce a **rewards system** to encourage citizens adoption of DynamicGuards, we would collaborate to local institutions to implement a rewards system, such as taxes reduction.

Collecting traffic data, PaveGuard could also provide insight for traffic patterns. The system may help optimize maintenance scheduling, prioritizing high-traffic areas to minimize disruptions.

In addiction to that, we think to make PaveGuard devices completely independent from external power sources thanks to solar panels and introduce air quality sensors on StaticGuards, in order to obtain relevant information about city air quality.

Second part

PaveGuard system: Overview

[FIL]

PaveGuard system, as already mentioned, is composed by many components.

StaticGuards and DynamicGuards use sensors to obtain telemetries from environment and exploit bridges to send them to backed over Internet.

Backend mainly performs two actions: receives telemetries from devices, storing them into database, and serves our clients.

Thanks to backend's API, the model and dashboard can get data. Model use data to perform predictions, dashboard visualizes them.

StaticGuard

[FIL]

StaticGuard device is placed on a road lamp, in order to capture traffic telemetries from above.

It uses a micro-controller and a set of sensors to collect information about:

- Temperature
- Humidity
- Rainfall
- Transits of its road

These information are sent using its bridge to remote server.

Bridge is integrated into micro-controller and it allows to establish a Wi-Fi connection.

Information about environment will be used to provide some historical data for all nearby streets.

DynamicGuard

[FIL]

TODO

Dashboard

[FIL]

TODO

Backend

[NIC]

Backend is the component that stores and manages information about entire system. It is composed by more remote modules that share data among them.

It stores information about users. There are two types of accounts:

- Administrators, who can analyze data and plan interventions
- Citizens, that are common accounts

Backend allows us to create digital representation of devices, acquiring and storing information about devices.

Each DynamicGuard is associated to citizen's account. StaticGuards don't have a owner and administrators can see their information using the dashboard.

In addiction, it manages data produced by devices and provides their telemetries through API, in order to allow other components (e.g. the model) to use them.

The Model

[NIC]

The Model is the game-changer component of our system. It exploits information about environment and actual pavements conditions, acquired by devices, to make predictions about future state of the roads.

The model is able to improve its performances, processing new data incoming from devices continuously, therefore it is able to operate in a dynamic environment.

The model provides predictions about future state of the roads, in order to allow administrators to plan targeted interventions.

Third part

In this next section we show more details about our system.

Development and Deployment

[NIC]

Every component of PaveGuard system was developed in an isolated environment thanks to Docker. Docker allows us to create isolated environments without the overhead of traditional virtual machines.

PaveGuard is an open source software, it is available on GitHub. Thanks to CI/CD, every time that a new version of a component is released, new code is deployed on remote servers automatically.

This ensure that users always use the last version of our softwares.

Backend

[NIC]

Backend is wrote using NestJS and stores data in a MongoDB database. It allows us to model entities of our system like devices and roads in a flexible way. We can obtain information about resources using its GraphQL APIs.

MongoDB is a NoSQL database picked to allow a more flexible management of different kind of telemetries. Telemetries are encoded as time series, in order to improve efficiency of operations. MongoDB cluster is used to ensure scalability and fault tolerance. It is deployed separately and remotely in cloud.

NestJS is a TypeScript framework used to build backend. It allows to build a solid and scalable APIs, which are used by every components in PaveGuard system to put new resources into database (e.g. telemetries) and retrieve information about entire system. Mongoose is used to handle MongoDB drivers and manage database operations.

GraphQL is a query language for APIs, we use it for all HTTP requests. The query language is a more user-friendly and descriptive alternative to RESTful APIs. We can easily develop new APIs thanks to resolvers and test them using the built-in APIs dashboard called Apollo sandbox.

StaticGuard

[NIC]

StaticGuard device is composed by a microcontroller which manages sensors. Firmware of StaticGuard was developed using Arduino libraries. Telemetries are sent to remote server using integrated bridge. StaticGuard is place above the road, in order to capture transits of vehicles of different shapes.

StaticGuard's firmware is executed on Arduino R4 Wi-Fi. This microcontroller allows us to manage sensors and build the device's bridge in an all-in-one board.

StaticGuard includes 3 types of sensors:

- DHT22 for temperature and humidity
- Rain Gauge to handle rainfall
- A pair of laser reflective sensors to obtain traffic data, in particular they are triggered when a vehicles transits below them

Bridge is integrated into device thanks to ESP32-S3 module of the Arduino board. ESP32 provides Wi-Fi connectivity, which is used to connect device to Internet.

StaticGuard 2

[NIC]

StaticGuard collects telemetries about temperature and humidity every 30 minutes checking a timer in each loop iteration. Instead, Rainfall and transits are handled through interrupt.

Temperature and humidity data from DHT22 are sampled thanks to DHT library provided by Adafruit. Instead, Rain gauge and laser reflective sensors have dry contacts which are directly connected to interruptable pins.

Rain gauge trigs a magnetic contact when approximately 0.3mm of water is stored into the bin.

StaticGuard detects a vehicle when it transits below it and both laser sensors are triggered in the right order.

Every measure generates a new telemetry. They are not sent instantaneously, but StaticBridge keeps in memory a bucket of telemetries, in order to optimize information sharing. When the bucket is too full, telemetries are encoded in GraphQL mutations and sent to backend.

In particular, telemetries are sent thanks to an HTTP POST request.

DynamicGuard

[FIL]

TODO

DynamicBridge

[FIL]

TODO

Data processing

[FIL]

TODO

The Model

[NIC]

The Model is internally composed by two regressor: the road cracks model and road potholes model. Each model predicts respectively future cracks and potholes severities.

The Model is re-trained periodically on processed data, in order to adapt predictions on dynamic environment.

We train the model aggregating all stored telemetries in process phase, in order to obtain enough records. Internal road cracks and potholes models are trained on respectively datasets, using a GridSearch of scikit-learn to find more suitable model type and hyper-parameters.

Surprisingly, we have noticed that linear models also obtain good performances, but currently we are using a decision tree.

To make predictions, every day we scrape roads in which there was a DynamicGuard transit. StaticGuards don't need to be present on those roads.

For each scraped road, thanks to Prophet we generate temperature, humidity and so on data of next 12 months of the nearby StaticGuards, in order to obtain processed data modulated by distance. Data preprocessing produces 2 features vectors which are used to predict road crack and pothole severities of next 12 months using pre-trained regressors.

Predictions are stored in remote database thanks to backend APIs, in order to provide them on dashboard and support interventions planning.

Dashboard

[FIL]

TODO