Design Project Report

STIJN DIJKSTRA, LUC DOP, TIM MULDER, and JUSTIN RUITER

Every day, many inspectors for Rijkswaterstaat go out to check the health of public infrastructure like bridges. This job is not simple and analyzing the data produced by an inspection takes time. With this project, we will investigate whether it is feasible to aid inspectors with an autonomously flying drone such as the DJI mini 2.

In this report we show that autonomous flight is possible on the DJI mini 2 using Live Mission Recording. However, we do recommend using a more expensive drone, but still a consumer drone. These drones usually have features like waypoint flight built in. Additionally, we built a web dashboard to show the user existing missions and results of our machine learning algorithm.

Authors' address: Stijn Dijkstra, s.j.dijkstra-2@student.utwente.nl; Luc Dop, l.t.j.dop@student.utwente.nl; Tim Mulder, t.t.mulder@student.utwente.nl; Justin Ruiter, j.a.g.ruiter@student.utwente.nl.

Contents

1 INTRODUCTION

In recent years, the drone market has seen significant growth with more types of drones becoming available to the public at affordable prices. While professional drones with features such as a 4K camera or a larger than 5 kilometer flight range still cost over €1000, it is projected that lightweight, hobbyist drones can be bought for under €100 [14]. These drones usually do not come with advanced software features such as Live Mission Recording (LMR). LMR is a way to record a set of waypoints previously flown by the drone to autonomously re-fly these exact waypoints in the future.

Our project involves building Live Mission Recording functionality in an Android app to control a DJI mini 2, a starter drone that costs around €500, having a weight of 249 grams, so anyone may fly it, even without a drone license. For this project, we worked together with TNO and Rijkswaterstaat, who are interested in the use case of recording a mission around critical infrastructure like bridges to automatically assess their health. These missions can later be reflown automatically to more easily build a library of pictures of the bridge at different times. The images will be stored on a server and analyzed by a machine learning algorithm.

Although we are building this project with TNO and Rijkswaterstaat, during the design and development, we made sure that every part of the system is easily replaceable if that is necessary. For example, the image processing pipeline is customized for Rijkswaterstaat, but it is a separate server and API so that it can be replaced by any API that follows the same specification if a different machine learning model is desired. Additionally, multiple machine learning models or image processing pipelines can be added to the server, so the image can get checked in many differeny ways.

2 DOMAIN ANALYSIS

In this section, we will discuss the domain in which our product will be used. We will explore the stakeholders and other involved parties in the use, or possible future use of our product and explore the current state of the art. In the end, we will emphasize the importance of this project, as we will describe the limitations of the current state of the art.

2.1 General Information

Our project can be used by anyone or any company that is interested in autonomously flying a pre-programmed route with a drone. Additionally, existing tasks which are currently not done by flying a drone could be replaced by a drone in the future. In fact, in the situation we are focusing on, that is the case. Rijkswaterstaat currently has to send a crew of inspectors to a bridge to inspect it. Although a drone might be used during this process currently, taking pictures of the bridge can be entirely automated in the future.

2.2 Involved Parties

During this project, we worked together with TNO, which works with Rijkswaterstaat to automate the task of inspecting vital infrastructure. TNO would like a solution that can control the drone and take pictures at a specific position, with a specific camera framing. Afterwards, this image should be automatically processed by machine learning, so Rijkswaterstaat can automatically receive the result of an inspection.

Our project supervisor is João Rebelo Moreira, who is interested in the IoT (Internet of Things) aspect of implementing such a project on a drone. He provided us with the data structure we should use for this project to easily interface with other IoT applications and devices: dronetology [8], although there were some limitations to this that we will describe later in the report. João also helped us integrate our project into Amazon AWS, a web

hosting platform frequently used by companies and by his chair. With these requirements, the project can easily be extended and modified in the future.

2.3 Existing Solutions

DJI already offers functionality like LMR and the automation of taking photos at a specific point with a specific camera angle, however, it reserves these features for its enterprise lineup of drones. Their DJI Pilot app includes all these functions [6] and has powerful functionality and even an online dashboard to manage missions and even a fleet of drones with [3].

For consumer-level drones, DJI offers a more basic SDK (Software Development Kit) which can be integrated with a custom Android or iOS application [4]. With this SDK the movement of the drone can be controlled, as well as the camera gimbal position. It also provides an easy way to extract the current location and state of the drone. It is not a direct solution to our problem, however, it can help us implement a solution.

The problem at hand is also not new. Drones are not new, although they have been becoming increasingly more accessible to the average consumer. In 2018, a paper was published investigating the feasibility of inspecting bridges using a drone, where they compared drones in many different price categories [9]. They concluded that although there are still many limitations, not all of which were imposed by the drone. This research, however, used the more expensive DJI Phantom 4 drone and concluded that cheaper consumer drones were less usable for the purpose we are attempting to use them for.

Because of the relative simplicity of detecting cracks in concrete for a machine learning algorithm, we chose to create a machine learning model based on cracks in concrete. This is meant as more of a demonstration of machine learning integration with this product, as fully inspecting a bridge by machine learning is out of scope for this project. It would require us building and labeling an extensive dataset of real bridges and testing it, which could be a project on its own. Detecting cracks in concrete has been done before. In their paper, Zwedu Ayele, et. al propose a method of mapping the photos onto a 3D model and determining damage based on that mapping [1]. This model was also used to more easily navigate the bridge itself and fly around its pillars. The individual images of various parts of the bridge are also processed and put through a machine learning algorithm. They concluded that this was just a proof of concept, but that it was not necessarily a tool to replace all bridge inspections with, although it might be possible and make bridge inspections safer and more efficient.

We believe that in order to make automatic infrastructure inspections by drone feasible, it should be possible to do these inspections with relatively affordable drones, as many of them are needed. So although the technology we are developing already exists on more expensive, enterprise drones, there is a good use for implementing it on cheaper drones. Combining this with image processing to make the technology feasible for deployment in real inspections has not yet been done before.

2.4 Limitations of Existing Solutions

Concluding our investigation of the current state of the art, although there are existing solutions on the market, they either use more expensive, enterprise, or near enterprise level drones or do not yet cover all our requirements. Additionally, although research has been done into parts of the solution for this problem, limitations were found. Usually, cheaper drones like the DJI mini 2 are not even considered for this purpose. This is because they are consumer-oriented drones. So, although what we are trying to achieve with this project has not

been done on consumer-oriented drones before, research into the feasibility has been done and not all results are encouraging. Processing the images, however, has been done before and should be feasible to implement on images made by our drone.

3 DESIGN PROCESS

In this section, we will describe the process for designing the different parts of the system. We will explain the design choices and how we incorporated the feedback we received into the final system design.

3.1 Preliminary Design

During the first few weeks of the project, we were mostly working on trying to understand the exact goals and requirements of the project. We had a meeting with our supervisor and TNO during this time. We explored the features of the drone and analysed the domain. After a few meetings, we understood what the research group and TNO wanted to see in our design. We started defining the requirements and user stories.

3.2 Initial Proposal

In week 4 we presented our initial proposal to our supervisor and TNO. They agreed with the general design but had a few additions to the requirements. They also suggested some changes to the interaction between the backend component and the classifier component. They requested that the classification of an image would be visible within the dashboard. During this meeting, we tried to immediately think of and propose some ways that these new requirements could be realised. We also discussed what possible downsides of the newly proposed solutions would be. This made it possible for the stakeholders to make an informed decision and together, we determined a few new requirements to work towards.

3.3 Incorporating Feedback

To make it possible to see the classification of an image in the dashboard, it was necessary to make a large change to the design concept. Our initial idea was that the communication with the classifier was one-way, the reason for this was flexibility. The classifier server would only need to be able to receive images. How and how quickly it handles those images was completely up to the implementation.

In the new design two-way communication is necessary, because the classifier server needs to send back the classification to the backend server. We decided to not have the classification be an immediate response to the request because this could cause issues with classifiers that are slower, or if the client wants the classification to be done manually. Therefore we designed the API in such a way that the classifier can make a new POST request containing the classification result once the classification process is done. We decided to store the classifications as strings so that both positive/negative and more complex classifications are possible.

4 USER STORIES

In this section, we will investigate the different actors and their requirements for our system.

4.1 Different Users

We split our users into three different types of users: The inspector, the system administrator and the client. The roles of these users will be expanded upon in their dedicated sections. After that, the user stories for each of these roles will be presented and explained.

4.1.1 Inspector. The inspector is the person who goes out into the field to record missions and deploy the drone when an inspection mission needs to be flown. He does not need access to the dashboard but only to the mobile app. The mobile app

can execute all his tasks, such as storing a recorded mission and loading an already stored mission.

4.1.2 System Administrator. The system administrator needs to have increased access to the system to manage existing missions and view the inspections that were done previously. In practice, the system administrator will most likely just be an inspector who is higher in seniority. In the dashboard, the administrator needs to be able to get an overview of all existing missions and delete missions as well. Additionally, all mission executions and photos associated with these executions need to be visible and the administrator needs to be able to rename individual waypoints in missions to more clearly mark where on the inspected structure the waypoint is located. Because of the DJI SDK not being able to properly retrieve images from the drone programmatically, our system will also allow the System Administrator to create a new mission execution and add photos to it in the dashboard.

Additionally, the system administrator should be able to determine how the images from the inspection are processed. For this, the system administrator must be able to change the classifier that a waypoint is classified with. This way, multiple aspects of the image can be assessed by different machine learning models and image processing pipelines.

4.1.3 Client. In a real-world implementation of our product, the client would most likely be a manager at Rijkswaterstaat, who can decide whether it is necessary to send inspectors to the bridge. Additionally, repairs could also be ordered based on the results of the inspection.

4.2 Inspector Stories

- As an inspector, I want to record a mission, so that I can execute it again later.
- As an inspector, I want to give newly recorded missions a name, so that it is clear to other users what the mission inspects.
- As an inspector, I want to execute an existing mission, so that I can collect inspection images.
- As an inspector, I want to sort available missions by location, so that I can choose the correct mission quicker.

4.3 System Administrator Stories

- As a system administrator, I want to change the name of a mission, so that it is clear to other users what the mission inspects.
- As a system administrator, I want to change the name of a waypoint, so that it is clear which part of the object this waypoint is meant to inspect.
- As a system administrator, I want to delete a mission, so that the system does not become cluttered with old missions.
- As a system administrator, I want to change the classifier of a waypoint, so that the system can easily change to an updated or a different image classification component.
- As a system administrator, I want to see the images taken on previous missions, so that I can manually determine if the system is working correctly.
- As a system administrator, I want to see how the images taken on a mission execution are classified and by what classifier.

4.4 Client Stories

- As a client, I want to receive damage reports, so that I can determine if infrastructure requires repairs.
- As a client, I want to be able to easily determine where on the inspected object the damage occurred.

5 REQUIREMENT ANALYSIS

5.1 Discovery Process

The requirements for this project originated from our stakeholders. Additionally, we also added some requirements ourselves, such as the non-functional performance requirements. Our main stakeholders were our project supervisor João Rebelo Moreira, TNO and by extension also Rijkswaterstaat.

João determined the basic requirements that were known before the beginning of this project together with Jurgis Šerkšnas, who investigated the feasibility of this project in his research project 'Implementing & Integrating Live Mission Recording with Starter Drones' [13]. We built upon his requirements and during the project, we continuously presented our progress and received updated requirements. We will list the final list of requirements we arrived at in this section.

Currently, TNO, a research organization in the Netherlands has a project with Rijkswaterstaat to automate the process of inspecting vital infrastructure such as bridges. Thus, TNO also added some requirements to the project. These were mainly for the dashboard and data storage part of the project, as they are interested in adapting our project to their systems. Although Rijkswaterstaat could provide us with some existing reports, they were not that useful for our purpose. They did include real inspection reports, but the image quality was often poor, the labelling of the images would need a lot of manual work and the dataset was not large enough to build a proper machine learning model upon. Because we did not have contact with representatives of Rijkswaterstaat directly, all requirements they had were added by TNO.

5.2 Functional Requirements

(1) The system must be able to save the state of the drone in a waypoint.
(2) A waypoint should consist of longitude, latitude, altitude, yaw, gimbal pitch, and the camera action.
(3) The system must be able to combine multiple waypoints into a replayable mission.
(4) The system must include a dashboard showing the existing missions.
(5) The system should provide an interface to store and retrieve mission information.
(6) The system must be able to store captured images.
(7) The system should provide an interface to store and retrieve images made during mission executions.
(8) The dashboard must include functionality to delete a mission.
(9) The dashboard must be able to display the images captured during previous missions.
(10) The mobile app should be able to suggest missions based on the current location of the drone and sort the missions on proximity to the users current location.
(11) The mobile app should be able to record and execute missions without requiring the use of the dashboard.
(12) The classifier component should run independently from the other components and be easily replaceable by a different model.
(13) The classification result should be stored in the database.

5.3 Non-functional Requirements

(14) When reaching the correct location, the drone should be able to rotate to the correct angle within 5 seconds.
(15) The system must be able to retrieve a waypoint from the API within 1 second (after the request has been sent to the API).
(16) The system must be able to store a waypoint within 1 second (after the request has been sent to the API).

(17) The machine learning application must be able to finish classifying an image in under 10 seconds.
(18) When a request is made to the online dashboard, the response should be sent within 2 seconds.
(19) The drone should be able to save at least 50 pictures per mission execution.
(20) Individual image size should not exceed 20MB.

6 ARCHITECTURAL DESIGN

6.1 Overview

As *figure 2* in the appendix shows, our system consists of four main components; an Android application that is responsible for controlling the drone, a dashboard that allows the user to view and manage missions and captured data, a classification server that can classify captured data, and a backend server to connect it all. These components all interact using HTTP requests.

6.2 Android Application and Drone Control

The Android application focuses solely on controlling the movements of the drone. It is built upon the three code repositories built by Jurgis Šerkšnas in his research project [13]. These repositories each provided part of the code required for flying and taking pictures with the drone autonomously. The first repository provides autonomous flight to a pre-programmed GPS coordinate and height [10], and the second repository provides control of the camera gimbal. This is done with the use of the MissionControl timeline controller that DJI provides with its SDK [11]. The third repository is a demonstration of how to store the current state of the drone in memory and includes storing the location, height, yaw angle, and gimbal position [12].

Our Android app is based upon the same DJI UX sample app the examples in the repositories are based on, and the app integrates the different functions to provide easy recording and playback of waypoints. The application can record waypoints and store a collection of waypoints as a mission in the API. Later, this mission can be chosen from a list of missions and reflown. When flying a mission, the drone can also take pictures, using the gimbal and yaw data to frame the picture as similarly as possible each time a mission is flown. The reason we opted to use Java and the DJI SDK for interfacing with the drone is because the code that was provided to us was already using these technologies. Apart from that, Java is a relatively simple language, we were already familiar with it, and the DJI SDK makes it possible for our application to work with many of DJI's drones, as Java is a language commonly used to build all kinds of mobile phone applications.

DJI provides a media manager with their SDK, which should make it simple for us to extract the pictures from the drone. However, we quickly discovered that it does not officially support the DJI mini 2, the specific drone model we used to develop this application. Sometimes, the media manager would report the correct list of files present on the SD card of the drone, but other times, it would only report one file that did not actually exist on the SD card. This problem is known by DJI and the community of developers using this SDK, as we could find multiple reports and Github issues of this problem specifically the DJI mini 2 [7]. In theory, other DJI drones should work fine. We solved this issue by moving this part of the upload process to the dashboard. In a real-world application, it is likely that a different drone model can be used and the last part of flying a mission can be added to the mobile app as well.

6.3 Backend System and Database

The backend of our system is highly focused on data. It functions as the data repository for all the

other components of our system. The backend consists of an Amazon RDS database and an Amazon EC2 instance. Amazon RDS is a service for hosting relational databases, which can be accessed by other AWS services or by external services. We use MySQL as our database engine because it is slightly less complex than database systems such as Oracle Database, while still offering enough features for our design. We favoured the combination of EC2 and RDS over AWS Amplify because the Amplify API system relies on GraphQL, which we were not yet familiar with prior to this project. We opted for a more traditional relational database approach using MySQL.

An EC2 instance is a compute instance capable of running various operating systems, which in our case runs Amazon Linux 2023. The instance functions as the bridge between the database and the other components of the system. On the EC2 instance is an HTTP web server that exposes API endpoints to the web. The web server is built using Apache Server and PHP. The reason we chose PHP is because it is lightweight and the initial setup is very quick. We did not use a PHP framework such as Laravel because we are not familiar with it and gaining familiarity would take too much time for relatively small benefits. Our API only has a few endpoints and is just a simple bridge between clients and the database, providing sanitization and authorization. The API offers complete read access through GET requests. To ensure data integrity the API limits the ability to directly modify data. Instead, the POST, PUT, and DELETE requests are handled by performing checks and actions that protect the database. For example, the */missionExecution* POST endpoint creates a new mission execution in the database and then creates the corresponding waypoint executions automatically. This design protects the data integrity by checking for undesired database states. It also makes it easier to debug errors by centralizing all the logic around data integrity in one place and being able to provide more informative status codes to the various clients.

The system uses the HTTP basic authentication header to authenticate a client. The credentials are in a configuration file on the server. For the connection between the RDS database and the EC2 instance, we use a feature in AWS, that automatically sets up the database and the instance so that the database only accepts connections from the EC2 instance. Using available enterprise solutions for critical components such as security is always a good choice when possible.

For compatibility reasons surrounding SSL certificates - which are necessary for secure traffic - we decided to use a proxy server. In our testing environment, we use a domain and hosting platform that Justin already had access to, because it was already available and simple to setup. Although we set up a self-signed certificate solution, in production, it is not a good idea to use a self signed certificate for securing internal connections. This is because the chain of trust is only limited to the certificate authority that we also control and is not as hardened as commercially available solutions like Let's Encrypt, or cloudflare. Currently, there is a self-signed SSL certificate on the web server, with a Let's Encrypt certificate to secure the outgoing connection from the proxy. In a production environment, it would be better to host the proxy on the same server as the API, as in that way, no unencrypted internet traffic leaves the EC2 instance.

The full documentation of the API is available at droneinc.stoplight.io/docs/dronelivemissionrecording.

6.4 Dashboard Web Application

To view current missions and edit their details, a dashboard was created. On the dashboard, the user can see a list of all the existing missions, and when a mission is clicked all waypoints of the mission and the previous executions of the mission

are shown. The waypoints are shown as clickable markers on a map, from here their name can and the classification model used to classify the image taken at this waypoint can be viewed. In a table, the details about the previous executions of the mission can be seen. From here, the user can view the pictures taken in a mission execution, delete a specific mission execution or create a new mission execution and upload its pictures. When the pictures are viewed, the dashboard also shows the user what model was used to classify each picture and how each picture was classified. Furthermore, entire missions can be deleted from the dashboard.

The dashboard is created as a React application, a JavaScript-based UI development library. We chose React since it is one of the easier libraries to use to create fast, efficient, and scalable web applications, and we already had some experience with it. This way we were able to create our own react components and combine these into the dashboard using CSS. The dashboard retrieves the data from the database by making several API calls. The dashboard sends GET requests to get information on all the existing missions and their details, as well as to retrieve the pictures from the database. New mission executions are sent to the database via POST requests, and the images are added to these executions via PUT requests. The removal of missions and mission executions is done using DELETE requests.

6.5 Image Recognition Server

To further automate the inspection of infrastructure, a machine learning model has been trained to detect damage in concrete. This model is then hosted on a web server that receives images and mission-specific identifiers and returns the predicted class of the image. The code for creating the model is written in Python 3.11. This is because of experience with the language, and because it has libraries for analyzing data and creating machine learning models. Python is a popular programming language for building Machine Leearning applications and has excellent documentation on how to create such applicatioins. The data used to train and test the model was reused from a different study [1]. Before feeding images into a computational model, they must first be simplified to reduce the computation required to process them. The first step is to resize the images to 80 by 80 pixels. Next is to grayscale the image. This means replacing the colors of an RGB image with differing shades of grey. Since the damage we are looking for does not rely on color this is a fitting way to decrease the complexity of an image. To do this scikit-image's rgb2gray function is used.

Afterwards, the image is transformed into a set of Histograms of Oriented Gradients (HOG) [2]. Local object appearance and shape in a picture can be characterized by the distribution of local intensity gradients or edge directions. This is done by first dividing the image into smaller regions or cells. Then for each cell, a one-dimensional histogram of gradient directions or edge orientations is created over the pixels of the cell. To enhance invariance to illumination, shadow, and edge contrast, contrast normalisation is performed. This entails grouping multiple cells into blocks and accumulating local histogram values. Cells typically appear in multiple blocks but the normalisations are block-dependent and thus different. This results in cells appearing multiple times in the final output vector with different normalisations. The normalized block descriptors are called HOG descriptors. These HOG descriptors are then collected from all blocks into a combined feature vector that can be used in a supervised learning model. To facilitate this process scikit-image's hog function is applied. For the transformation in this model, cells have a size of 14 by 14 pixels with a block having a size of 2 by 2 cells.

The final feature factor we receive after the transformation has a high cardinality. For this reason, we have opted to use a support vector machine

(SVM). An SVM is a supervised learning method provided by sci-kit learn which can be used to classify inputs. It is versatile and effective in handling high-dimensional spaces such as our feature vector. It is even effective when the number of features is greater than the number of samples, however, this can lead to over-fitting. Another downside is the SVM being unable to provide a direct probability estimate, but this can still be attained with a computationally expensive 5-fold cross-validation. To decrease the amount of computation required in training the SVM uses Stochastic Gradient Descent (SGD) learning. Here the model updates its internal parameters after each processed sample with a decreasing learning rate. The model is supplied with a dataset of 20,000 JPEG images, where exactly half shows cracked concrete and the other half shows non-cracked concrete. This dataset is split into an 80/20 distribution of training and test data respectively. To increase the model's accuracy hyper-parameter tuning is performed. This involves trying a variety of different values for the model's hyper-parameters and taking the values that yield the highest accuracy score. The model is now ready to be used. It is saved as a .pkl file to be loaded and deployed on the server.

To create the web application and process classification requests the Python library flask is used. This allows HTTP requests to be routed to the correct functions. The web app is protected with basic authentication, just like the API. On startup, the web app loads the locally stored model. It is then able to receive classification requests in the form of a POST request containing a base64 encoded image and identifiers. The result of the classification along with the identifiers is then sent to the database in the form of a PUT request. To allow the web app to communicate with the web server it is hosted on, Gunicorn is used as a Web Server Gateway Interface (WSGI). The HTTP server capabilities are provided by NGINX, which acts as a reverse proxy. All these processes are run on Ubuntu 22.04, which is hosted on an Amazon AWS EC2 instance.

6.6 Flexibility

Our system is designed with flexibility in mind. This can mainly be seen in the connection between the API web server and the machine learning server. Our API can handle multiple different machine learning algorithms and a different machine learning algorithm can be chosen for each mission. It is even possible to choose a different machine learning model for each waypoint in a mission. For example, one mission could check both concrete degradation and corrosion on metal parts using two different classifiers. If the machine learning algorithm follows the API specification, it should be possible to seamlessly swap between machine learning servers. The system is so flexible, that the classification could even be done manually because there are no timing constraints. In our project, we developed a basic implementation of a machine learning algorithm that can detect cracks in concrete as a proof of concept.

In a real deployment by Rijkswaterstaat, it would likely be necessary to run the images through multiple different image processing pipelines and machine learning algorithms. This, however, was not feasible to implement in our project in the time allocated to it. Additionally, TNO could not provide us with any usable dataset for achieving this goal, so a dataset to train these algorithms on would most likely also need to be built. This task is left up to future research, or TNO and Rijkswaterstaat, if this project were to be tested in a real inspection scenario.

7 USER INTERFACE OVERVIEW

7.1 Features

The features in the dashboard are created so a system administrator can view and edit the existing missions. On the left side of the dashboard, a list can be seen containing all the current missions.

The missions are displayed in simple blocks listing their mission id, the name of the mission and information on the number of waypoints and photo waypoints to make a better distinction between different missions. The mission blocks can be clicked to display more detailed information on the right side of the dashboard. Here, an overview of the waypoints in the mission is given in the form of a map with markers. Also, on the bottom of the screen, a table is shown with information on the previous executions of this mission. Here, the date of the executions is shown, as well as an option to view the pictures taken during the execution as well as an option to delete the execution. When viewing the pictures, the dashboard also shows how the picture is classified and what classification model was used. On the top of the screen, a new mission execution can be created and the mission itself can be deleted.

In the mobile app, we kept the features basic by design. As the app is used in the field, it is not necessary to have advanced functionality like viewing previous mission executions. During flight, the user can create a mission by recording waypoints of the drones current position and state. The drone will only remember the waypoints themselves and not how the user flew to it, to make it possible to fly the optimal path between waypoints. The user can send a collection of waypoints to the API after giving the mission a name or reset the currently saved waypoints. When an inspection needs to be done, the user can select from a list of waypoints retrieved from the API, sorted by the proximity of the first waypoint to the user. The app loads the waypoints in the mission into its memory, from where the user can start a mission execution. Because of the limitations in the DJI SDK described previously, the app does not start a mission execution and upload the images taken during the execution. This needs to be done manually, in the dashboard after flying the mission.

7.2 Usability

The design of the dashboard is kept very basic for this project. After discussing with our supervisor, João, we concluded that we wanted the dashboard to be a proof of concept that shows the intended functionality without an elaborate user interface, so we would not spend too much time on this. In *Figure 9* in the Appendix, the main page of the dashboard can be seen. On the left, the missions list is shown that sorts the missions by their mission ID. The mission details section on the right side will simply display placeholder data until a mission is clicked from the mission list. When viewing the pictures taken in a specific mission execution, the dashboard shows them like is shown in *Figure 10*.

The features the app supports are basic by design, as explained in the previous section. Different functionality is put under different buttons that pop up individual menus for managing waypoints, or missions. If something goes wrong, the user can press the emergency stop button and the drone will stay where it is and stop flying and freeze on the spot it was flying through when the emergency stop button was pressed. In *Figure 7* the main screen of the drone can be found. Under missions and load a mission from the API, the user can load a mission from the API to fly it from memory. This screen sorts the missions by proximity to the current location of the drone and can be seen in *Figure 8*. The proximity to a mission is determined by the first waypoint in the mission and the app limits users by only allowing the drone to fly to waypoints within a 500 meter radius from the drone's current position. This prevents the user from accidentally flying the wrong mission and reduces the chance that the drone will crash into something. For the best experience, the user should deploy the drone at the exact same location every time, or know where the drone will fly so the drone does not accidentally crash into something.

7.3 Considerations in Further Design

As mentioned in section 7.2, we did not spend much time working on the user interface and making the dashboard aesthetically pleasing. We suspect that if this product were deployed by a client, they would want to create their own dashboard for easier integration with their systems, or integrate the dashboard fully in existing systems. In this case, they might take the functionality of our dashboard as a starting point, but they could definitely improve on the user interface and general looks and feel of the dashboard. Furthermore, the dashboard might be improved upon by adding more functionality to edit the missions. For example, a system administrator might want to edit the location of waypoints to fine-tune a mission, or delete certain waypoints in a mission that might no longer be possible to reach.

As for the mobile app, because our product is more of a proof of concept and we were adapting an app with a codebase that we were not familiar with, the final product is not very aesthetically pleasing. This is not such a big issue for deploying it with a company, as the UI is intuitive and all the functionality is implemented. However, in future work, it might be possible to improve the UI and interact with the camera feed of the drone as well to show the waypoints visually, through the camera of the drone. Additionally, there is already a basic implementation of a map in the example DJI provides, which can be expanded upon to show where the waypoints are on the map. We did not do that however, as such functionality is already implemented in the dashboard. Streamlining the process of flying a mission so it takes place entirely in the app (including creating a mission execution in the API and uploading the photos of the mission execution to the API) would also make the process less error-prone.

8 TESTING AND VALIDATION

8.1 Testing Approach

Our testing approach mostly revolves around manual testing and integration testing. The individual components of our system are relatively small and are all structured in different ways using different languages. This makes it difficult to create a structured testing approach. Testing the drone can only be done manually. Luckily, the small scope of the individual components makes manual testing feasible. After each change, we tested the affected endpoints. After large structural changes, we retested the whole API.

8.2 Integration Testing

In the last two weeks of development (weeks 7 and 8), we focused on integrating all the different components of our system. Integrating the drone with the backend component was done at an earlier stage. At each stage of integration, we tested if everything worked as expected. We started by integrating the machine learning component with the backend, then tested it by manually uploading images to the backend component to see if it would correctly do the two-way communication with the classifier. Next, we checked if data uploaded using the dashboard would also correctly go through each stage and store a classification in the database. Then we also confirmed that the classification could be retrieved by the dashboard.

8.3 Requirement Validation

The functional requirements can be easily tested by using the functions of the system while inspecting the behavior of the backend at the same time. This way we could ensure that all the data that is sent to the backend is stored in the way we would expect. The non-functional requirements are evaluated by looking at the achieved performance of our system. Validating these requirements was done at the same time as integration testing. We measured performance in multiple situations by creating test

missions that focused on specific characteristics, such as turning speed or long-distance flight precision. After testing these situations we can conclude that our system adheres to the non-functional requirements.

We specifically tested the following situations:

- We created a mission where a waypoint is far away from the previous. Using this we can test if the drone correctly maintains its altitude while flying.
- We created a mission where the drone has to make a sharp turn. Using this we can test how quickly the drone turns.
- We created a mission where the drone rapidly drops in altitude, to see if it does not drop lower than the target altitude, or if it would correct itself.

8.4 Machine learning model analysis

To determine how well the machine learning model performs we have performed an analysis of its predictions. As mentioned in section 6.5, the 20.000 images made available to the model are split into an 80/20 distribution. This leaves 4.000 images for testing. The model then labels these images 'Posi' if it predicts an image to be of cracked concrete, and 'Nega' if it predicts the image to be of undamaged concrete. These predictions are then compared to the actual labels of these images. We have gathered the results of this in the confusion matrix seen in *figure 11*. What is interesting to note is that the model seems to perform quite well with the test data, scoring 98.1% accuracy in its predictions. To determine if the model can adapt to new, unseen data a 5-fold cross-validation is performed. Here we split the data into 5 folds of different 80/20 distributed training and test sets and examine how the model performs on each fold. We measure its performance based on the following metrics:

$$Accuracy(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} 1(y_i = \hat{y}_i) \tag{1}$$

$$Precision(y, \hat{y}) = \frac{|y \cap \hat{y}|}{\hat{y}} \tag{2}$$

$$Recall(y, \hat{y}) = \frac{|y \cap \hat{y}|}{y} \tag{3}$$

$$F1(y, \hat{y}) = 2 * \frac{Precision(y, \hat{y}) * Recall(y, \hat{y})}{Precision(y, \hat{y}) + Recall(y, \hat{y})} \tag{4}$$

Here n is the number of samples, $\hat{y}$ are the predicted values, and y are the corresponding true values. 1(x) is the indicator function. For each fold, we take the scores of the metrics and calculate the mean result. A graph showing the mean scores of all folds can be seen in *figure 12*. What stands out about this graph is that all scores are around 0.975. This can be explained when looking at the confusion matrix in *figure 11*. This shows that the number of false positives and false negatives is very low and around the same number. When this is the case for every fold in the 5-fold cross-validation, this results in very similar mean scores. This shows that the model can generalize when faced with unknown data, and is therefore not overfitted or underfitted. While the model is not overfitted or underfitted, it is important to analyze if the amount of training data is optimal. To this end, we have created a learning curve as seen in *figure 13*. This involves training the model with increasingly bigger subsets of our actual training data, and examining how its accuracy changes. What we see is that while the model benefits from a large dataset, the actual increase in accuracy is limited. A training set with over 12.000 images only performs around 2% better than a dataset with less than 2.000 images. An explanation for this could be the way the model learns when presented with increasing amounts of data. As mentioned in section 6.5, SDG learning is used which causes the model to learn at a slower rate when given more data. Other learning algorithms could lead to different results and should be examined in further research.

9 REFLECTION

9.1 Planning

At the beginning of the project, we made a plan to ensure that we would finish every feature in time. We also determined dependencies between features and different parts of the system, so we would be able to actually follow the planning. Our initial Gantt chart can be found in *Figure 1*.

This Gantt chart is based on the requirements we had at the time of creating it. We expected that, through our collaboration with TNO, additional requirements would be added. We also expected issues with integrating the autonomous flight code that was provided to us, which is why some of the implementation tasks are planned to take longer than may be expected.

In the end, we needed the extra time in this Gantt chart, because during the integration of the three repositories provided to us, we had a lot of issues, ranging from bugs in the DJI SDK itself, as well as concurrency issues. Because of the uncertainty in our collaboration with TNO, we opted not to create a new Gantt chart after this. We made a plan each week, after our weekly progress meeting with our supervisor, based on the feedback discussed in these sessions. This worked well for us.

9.2 Cooperation

In the initial phases of the project we made some agreements on how we would cooperate during the development of this project. One thing we deemed important was frequent in-person work sessions, as this would keep members up-to-date on proceedings and foster collaboration. Another agreement was an equal distribution of the workload. We did not think it necessary to formalize these agreements in a written contract as we hold ourselves to be responsible enough to uphold them. We would say that agreements were properly followed by all members of the team throughout the duration of this project and that any deviations from them were reasonable and/or discussed beforehand. The Red/Green card procedure was not relevant to our project, as we feel every member of the group contributed evenly to its progress.

9.3 Stakeholder Interactions

During the development of the project, we tried to have frequent contact with the primary stakeholders. For this, we planned a meeting every Friday to discuss our project with the stakeholders. Beyond these meetings, the main methods of communication were via Microsoft Teams and e-mail. Throughout the development of the project our supervisor, dr. João L.R. Moreira, proved to be a motivated source of feedback. Though initially, his attention had to be divided because of personal circumstances, he was readily available to answer any questions we had. The contact with our other primary stakeholder, TNO, was a bit unclear at the start. Our initial expectation was that TNO would be able to offer us datasets to train our machine learning model on, but this was not the case. They instead acted more as a client, in that role they offered valuable feedback on our design and had critical questions about how it would function in practice. We are not sure where the initial misunderstanding came from, but in the end, their professional knowledge of the subject of infrastructure inspection was very helpful during our project.

9.4 Issues

During the design and development of this project, we had many issues, from issues with the DJI SDK and the drone to the weather and the technology provided to us. In this section, we will describe these issues and the solutions we found for them, if any.

Starting with the drone and the DJI SDK: DJI has released a new SDK version, version 5.0 and the documentation and questions we found online did

not necessarily specify for which of the SDK versions they were. We are using 4.16.2 in our project, which is one of the newest versions of SDK version 4. The reason we cannot switch to SDK version 5 yet, is that the DJI mini 2 (the only drone available to us for this project) is not supported by that version anymore [5]. Additionally, the MediaManager component of SDK version 4 does officially not support the drone we are using for development [7]. This will also not be fixed and might be the result of DJI giving a lower priority because of the lower price of the drone, although we cannot say this for sure.

Flying the drone relies on a nice open space being available. During development, we could not always ensure that the drone would do exactly what we thought we programmed it to do. We used the UTrack for testing, as there was a nice open space available in which we could safely test our code. However, as the drone is not waterproof, we also needed to plan around weather forecasts. Because of GPS reception, it was impossible for us to test the drone inside a large indoor space during days with bad weather, so some days we could just not make any progress on testing the Android application.

Amazon AWS also posed some issues for us. Using AWS as infrastructure to deploy and host the systems of our project was a requirement, however, we were not familiar with AWS before this project. Thus, we opted to just use the basics of AWS (an EC2 instance, which is just like a standard Linux server) and an RDS instance to create and host the database. For the dashboard, we used an Amplify instance. We had some small starting problems like the EC2 instance not having a static IP address and a bug in Visual Studio code that caused the instance to hang every time we tried to update something on it.

10 CONCLUSION

10.1 Thoughts on DJI SDK

Currently, there are two different versions of the DJI SDK: version 4 and version 5. Version 4 supports older drones and receives less updates than the newer version 5. In our project, we used the DJI mini 2 drone, which was only supported by SDK version 4. For future projects, we would recommend that if the drone supports it, SDK version 5 is used.

Although we managed to work our way around the SDK, on forums on the internet, it is not always clear which version of the SDK the conversation is about. This makes it often difficult to effectively find a solution to a problem. Additionally, the SDK would log a lot of messages in the console, as well as warnings or errors. Although these did not seem to impact the working of our app too much, this is an example of a larger issue. This issue is that the inner workings of the SDK were not always clear to us and it was often difficult to diagnose where issues were coming from. With regard to concurrency, we had more issues. Some functions and structures did not seem to have a proper callback structure, or callbacks that were not suited for detecting when they were finished. We had to build around these limitations, sometimes making assumptions about the time an action would take, which is clearly not the desired solution.

Apart from these issues, the DJI SDK makes it trivial to implement a solution that will support many different models of DJI drones. If many different models of drones need to be supported, the DJI SDK seems to be the way to go. Because we did not have access to a drone that is supported by DJI SDK version 5, we could not test that version of the SDK specifically and it might have solved some of the issues we encountered with version 4.

10.2 Feasibility of Consumer Drones

This project has shown that although there might be some limitations, using consumer drones for automating flight is definitely feasible on a small scale. The code we developed is currently stable and we can safely fly it in a crowded area where GPS reception is good. This might however not be the case for bridges, as a mass of concrete or metal might interfere with the GPS signal. For bridges, it might be possible to install specific hardware on the bridge which the drone can recognize and use to accurately determine its location in relation to the bridge.

Higher-end drones like the DJI mini 4 pro, have, as previously described, more of the features required to carry out automated inspections of bridges, like built-in waypoint support, which can be interacted with with the SDK as well. Additionally, these more expensive drones also have object detection in a 360-degree radius around them, preventing a fly-in with a bridge if the GPS signal does get lost. It might thus be worth it to use a slightly more expensive drone.

10.3 Documentation

We documented our code and system in a few different ways. Our entire system is mapped by a full architecture overview and the API is documented by an openAPI documentation. This documentation maps out every endpoints, arguments and gives examples on how to use every endpoint. Additionally, we were requested to implement dronetology [8]. This ontology is clearly in its early phase of development and although it is very extensive, explanations for many terms and properties are lacking. Additionally, dronetology seems to be more useful for documenting an entire system, from drone to data storage, whereas we just need to format our API according to dronetology. We chose to try to name our properties according to how dronetology prescribes it, but we could not implement dronetology fully, as its documentation was lacking. Improving this could be a project for future work.

10.4 Recommendations

The DJI mini 2 is a great starter drone. In drone terms, it is not very expensive, it has a camera that is good enough for taking nice pictures from above and can even fly decently fast. However, we also had a few issues with it, which are described in the issues section. Its relatively cheap price might also incur some of these issues, although we cannot say that for sure. Although we tested and developed with the DJI mini 2, the great thing about the DJI SDK is that our code will work with more drones than just the DJI mini 2. In this section, we will make a recommendation for a drone to be used if this project were to be continued with a different drone.

First, it is important to consider the DJI SDK version the drone is supported by. Although active development on version 4 seems to have just ended, no new drones are receiving support for this version and thus, to ensure the drone keeps receiving SDK support, it would be wise to choose a drone that is supported by SDK version 5. We have personally tested the DJI mini 4 Pro. The main benefit of this drone is the controller. It connects with the DJI RC controller version 2, which runs Android as its operating system. Although it might not be trivial and we could not test it with this particular drone, it might be possible to deploy an app directly to this controller and cut out the requirement for having a connection to the controller.

A disadvantage of choosing the DJI mini 4 pro is its price. Although with its price under €1000.00, it can still be considered a starter drone, it is twice as expensive as the DJI mini 2. However, with the DJI mini 2, a phone would need to be purchased as well and integrating everything onto the controller

directly might also lead to a more stable and user-friendly system. Additionally, the DJI mini 4 pro also supports waypoint flight out of the box, which can also be accessed by the SDK. This would make for a more seamless integration process.

11 FUTURE WORK

11.1 Android Device Emulation

One limitation of using the DJI SDK is that it requires the controller to be connected to an Android device. That means that one device can only control one drone and that there is no USB port available for debugging information and charging. For real-world applications, it would be beneficial to have a system that is capable of emulating multiple Android devices and has a central information log to keep track of the multiple drones. As noted in the recommendations, newer DJI drones have controllers that run Android and don't require an additional device. It may be possible to reverse-engineer those new controllers to help build a system that can directly control newer drone models. Even without the need for Android emulation.

11.2 Improvement of Image Classification Models

A limitation of the current image classification model is that it is not trained using images made by the DJI mini 2 or similar drones. This means that while it does perform well on the training data, it is unsatisfactory when dealing with actual pictures. Though it was unfeasible in the limited time we had for the project, in further development it would be good to create a dataset solely for this project. Further improvements to the image classification model could come in the form of creating multiple models for different kinds of infrastructure damage. As it stands the model is only suited for detecting cracks in concrete, but damage to infrastructure could take a variety of forms. Further research should be done into what kinds of visible damage can occur in infrastructure and make separate models for each.

REFERENCES

[1] Yonas Zewdu Ayele, Mostafa Aliyari, David Griffths, and Enrique Lopez Droguett. 2020. Automatic crack segmentation for uav-assisted bridge inspection. *Energies* 13, 23 (12 2020). https://doi.org/10.3390/EN13236250

[2] N Dalal and B Triggs. 2005. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Vol. 1. 886–893. https://doi.org/10.1109/CVPR.2005.177

[3] DJI. [n. d.]. DJI FlightHub 2 - Drone Software and Management Platform - DJI Enterprise. https://enterprise.dji.com/flighthub-2

[4] DJI. 2022. GitHub - dji-sdk/Mobile-SDK-Android: DJI Mobile SDK for Android: http://developer.dji.com/mobile-sdk/. https://github.com/dji-sdk/Mobile-SDK-Android

[5] DJI-SDK. 2023. dji-sdk/Mobile-SDK-Android-V5: MSDK V5 Sample. https://github.com/dji-sdk/Mobile-SDK-Android-V5

[6] DJI Ennterprise. 2020. Smart Inspection (Waypoint 2.0, Live Mission Recording, AI Spot-Check), Matrice 300RTK & H20 Series. https://www.youtube.com/watch?v=XYoBx1As6Sg

[7] gianlu33. 2022. Mini 2: cannot retrieve media list with 'getSDCardFileListSnapshot' · Issue #1188 · dji-sdk/Mobile-SDK-Android. https://github.com/dji-sdk/Mobile-SDK-Android/issues/1188

[8] David Martín-Lammerding, Alberto Córdoba, Jesús Villadangos, and José Javier Astrain. [n. d.]. Dronetology. https://dronetology.net/dronetology/index-en.html

[9] Junwon Seo, Luis Duque, and Jim Wacker. 2018. Drone-enabled bridge inspection methodology and application. *Automation in Construction* 94 (10 2018), 112–126. https://doi.org/10.1016/J.AUTCON.2018.06.006

[10] Jurgis Šerkšnas. 2023. GitHub - LarvaZZa/LMR-Autonomous-R1. https://github.com/LarvaZZa/LMR-Autonomous-R1

[11] Jurgis Šerkšnas. 2023. GitHub - LarvaZZa/LMR-Inspection-R2. https://github.com/LarvaZZa/LMR-Inspection-R2

[12] Jurgis Šerkšnas. 2023. GitHub - LarvaZZa/LMR-State-R3. https://github.com/LarvaZZa/LMR-State-R3

[13] Jurgis Šerkšnas. 2023. *Implementing & Integrating Live Mission Recording with Starter Drones.* Ph. D. Dissertation. University of Twente, Enschede.

[14] Statistica. 2022. Drones - Worldwide | Statista Market Forecast. https://www.statista.com/outlook/cmo/consumer-electronics/drones/worldwide

A GANTT CHART

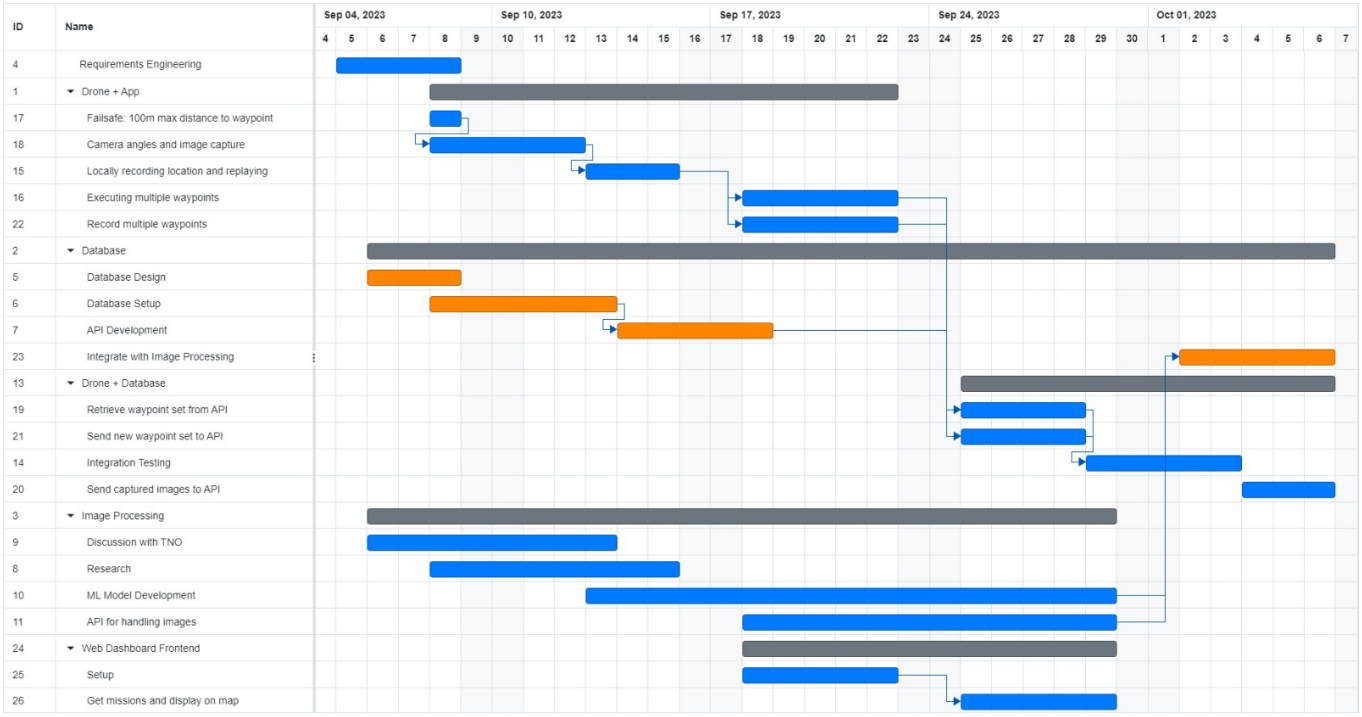

Fig. 1. The Gantt chart we made at the beginning of the project to streamline the first few weeks of the project

B DIAGRAMS

B.1 Architecture Overview

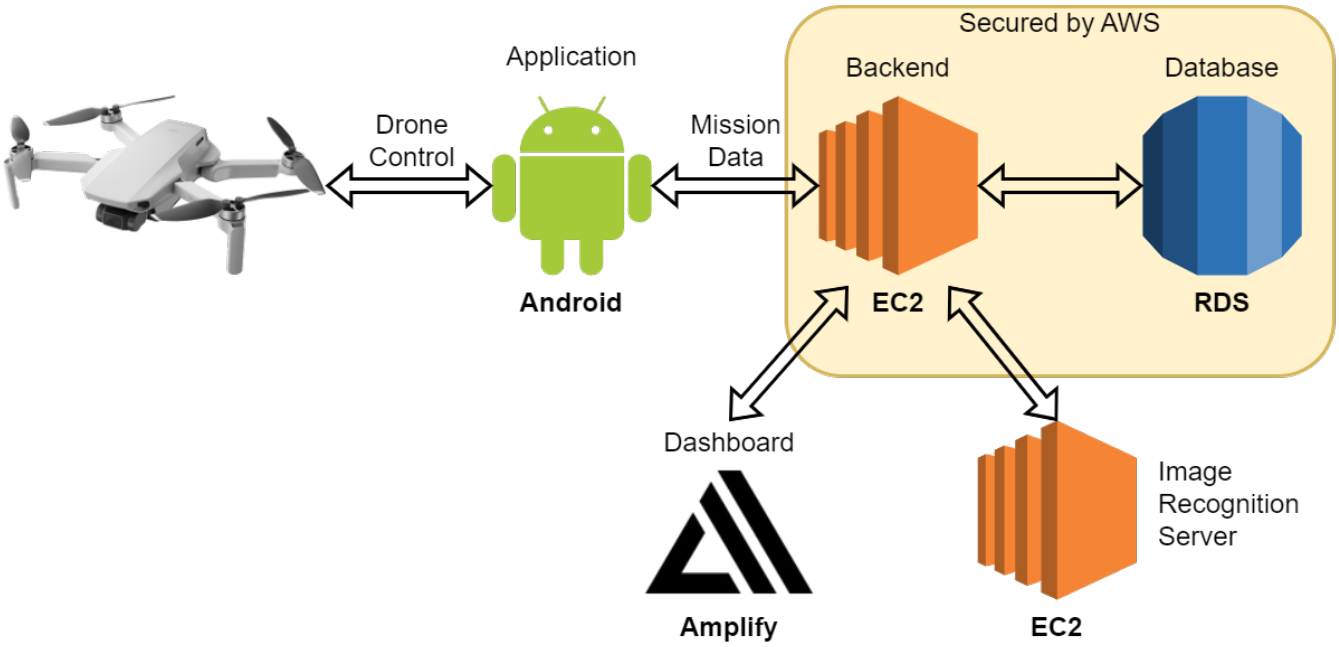

Fig. 2. Overview of the components of the architecture and the communication between them

B.2 Database diagram

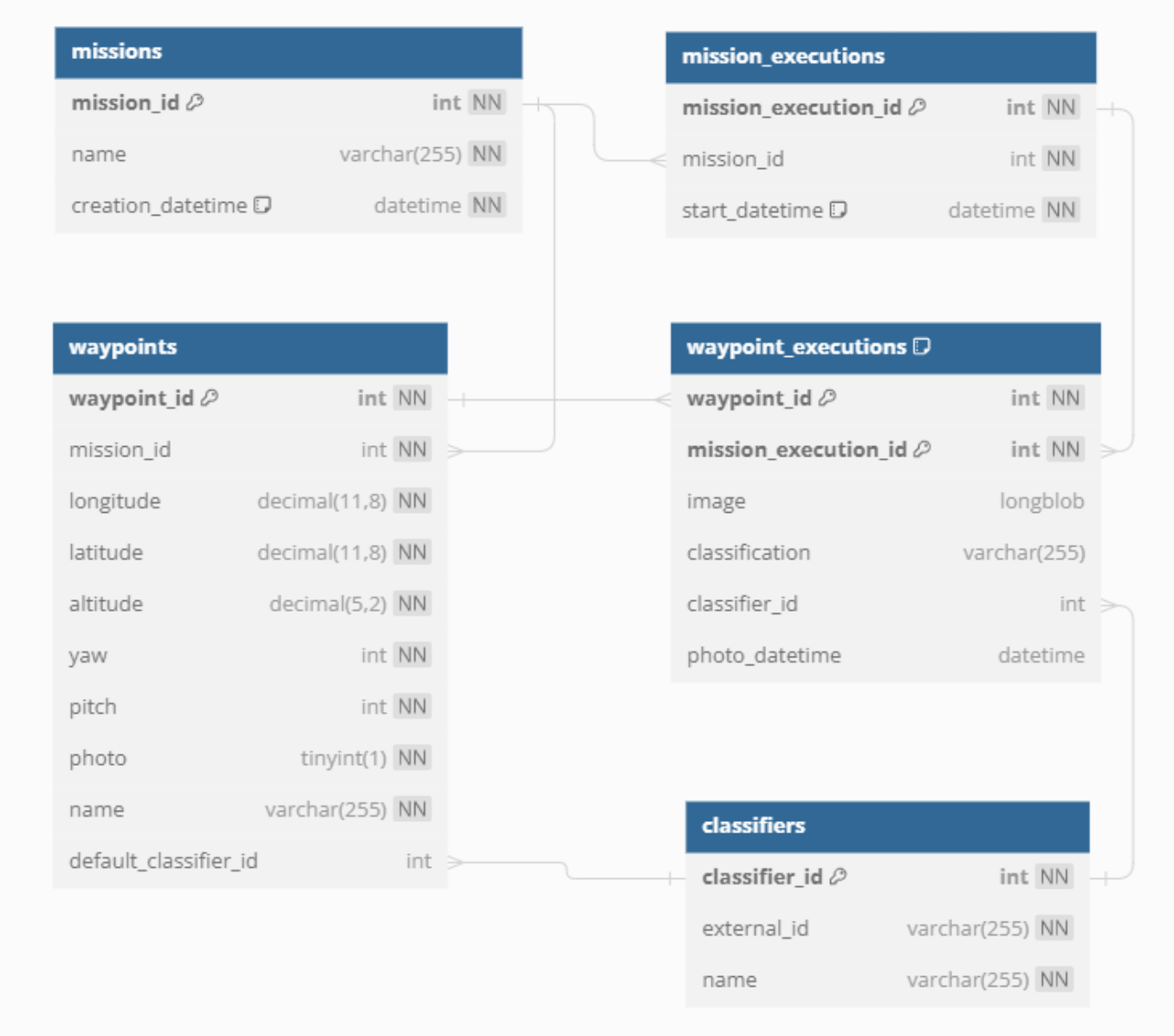

Fig. 3. The database structure

B.3 Activity diagrams

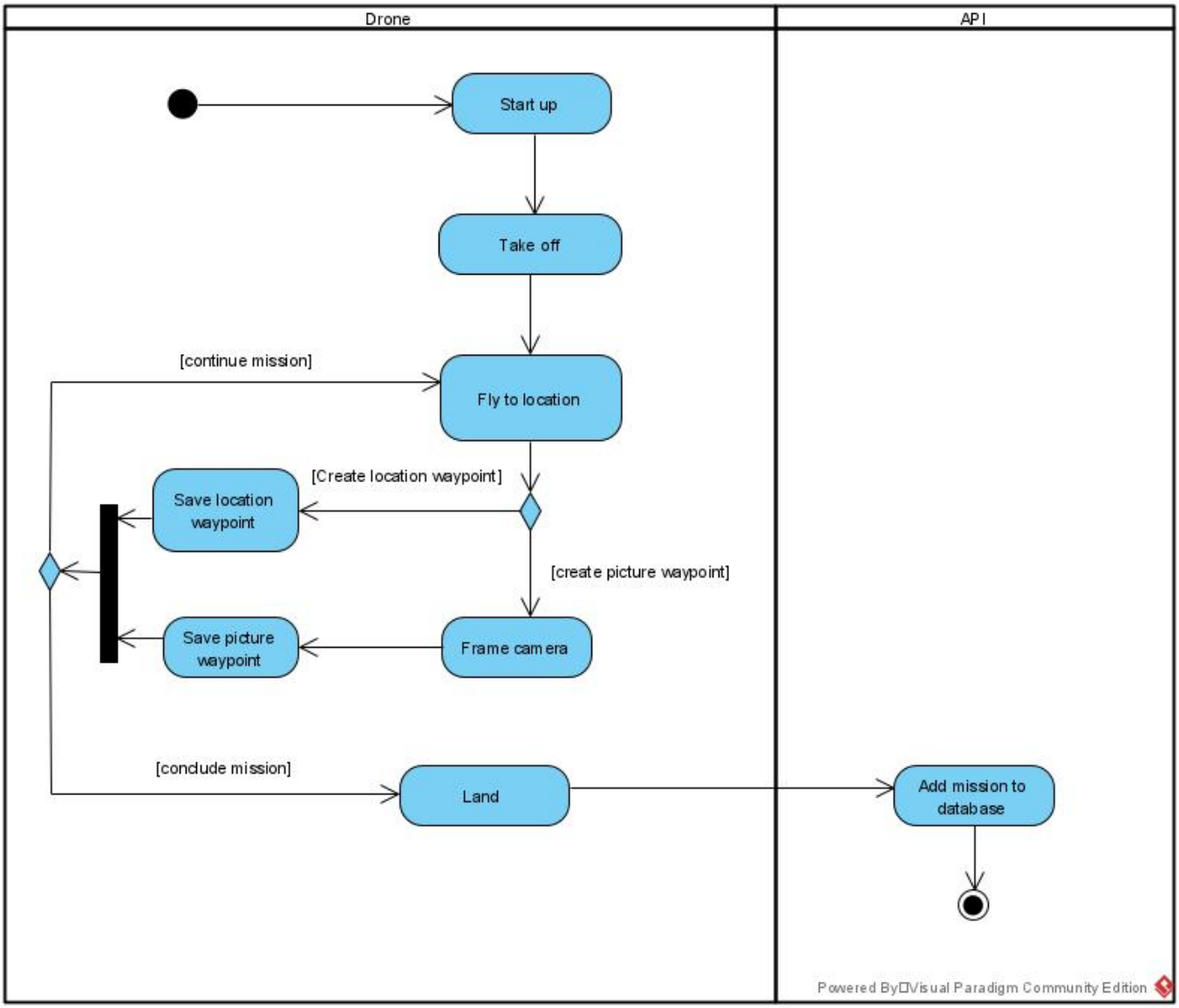

Fig. 4. Activity diagram for recording a new mission

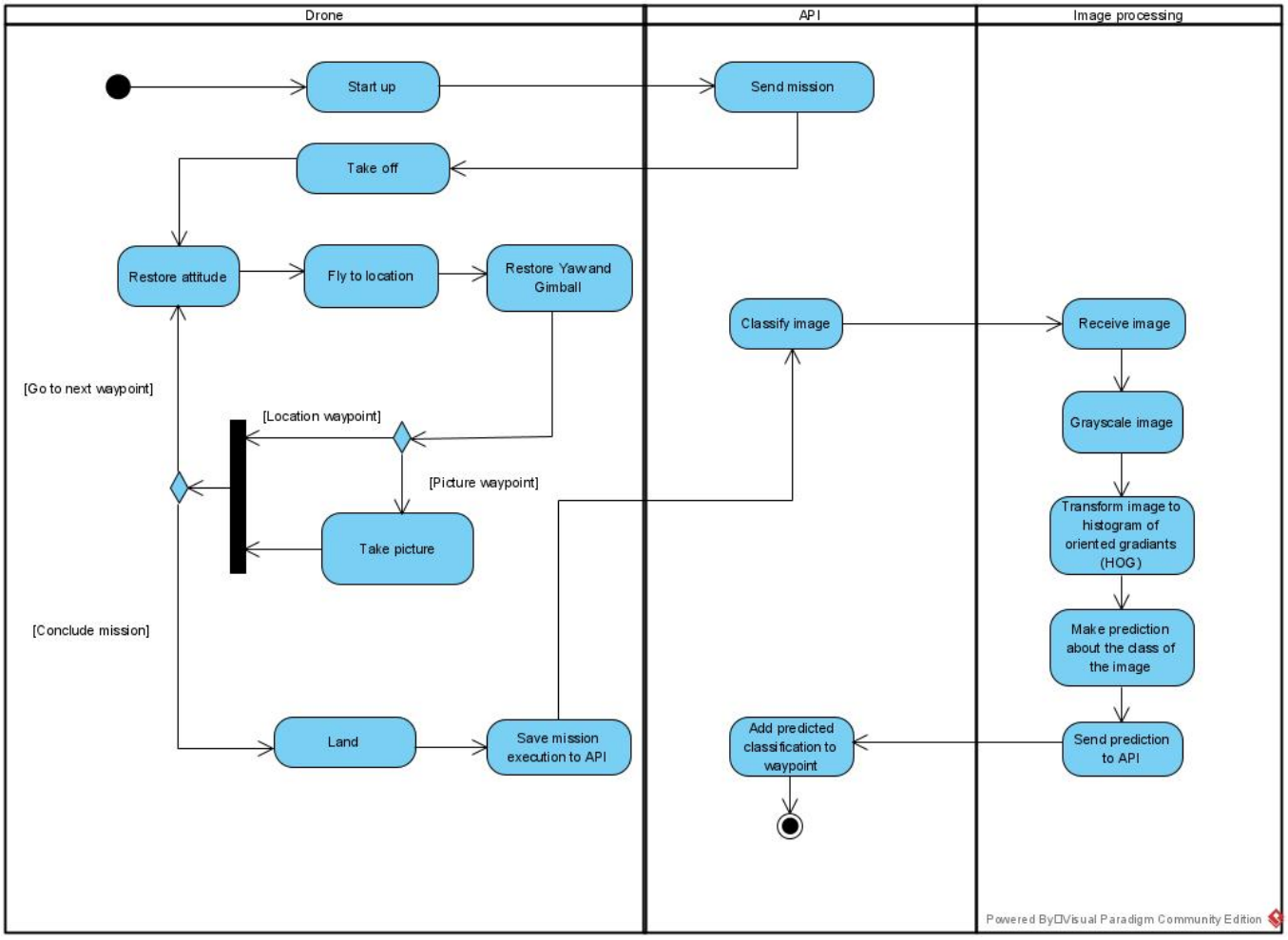

Fig. 5. Activity diagram for loading an existing mission

B.4 Use case diagram

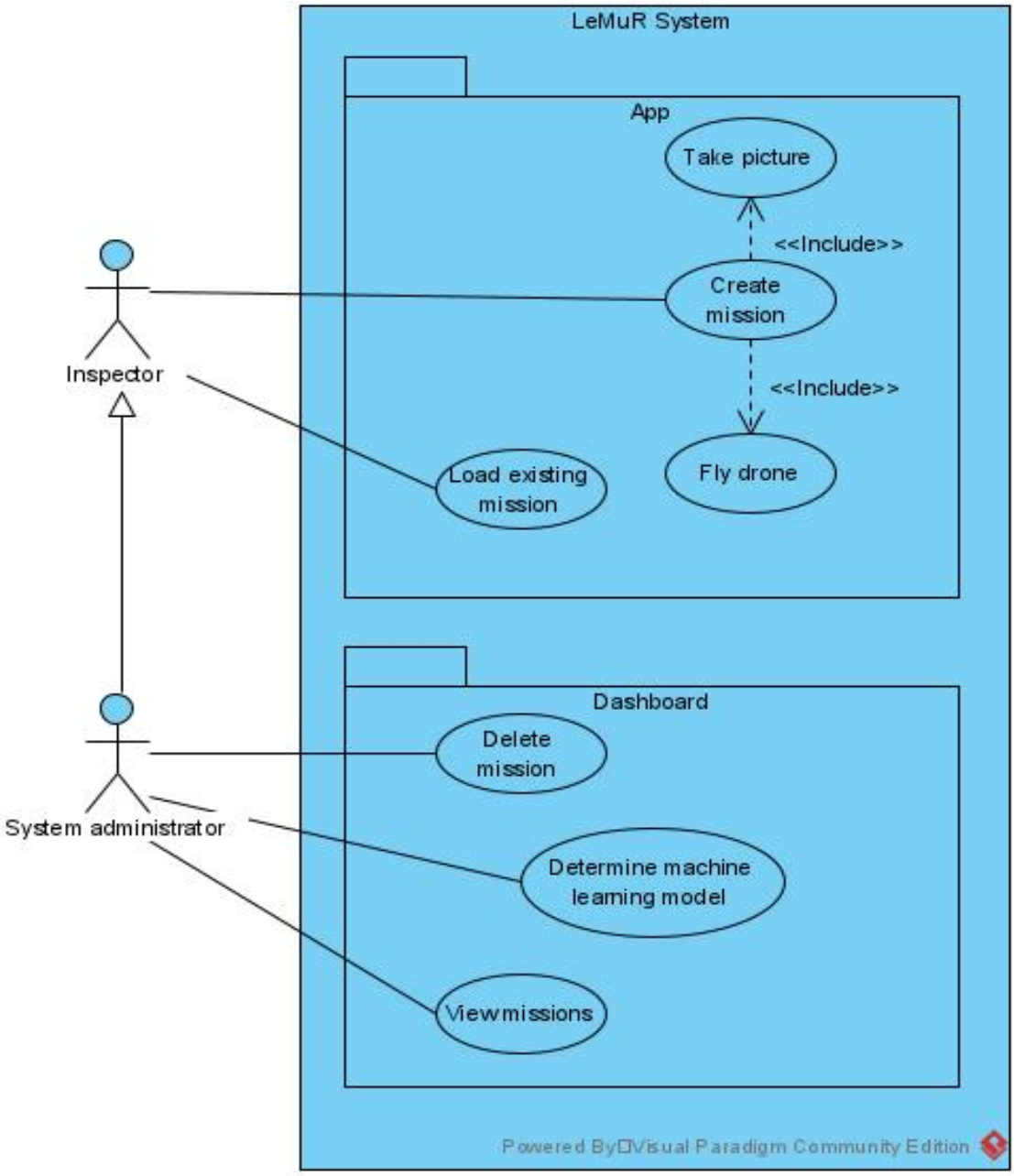

Fig. 6. Use case diagram

C MOBILE APPLICATION

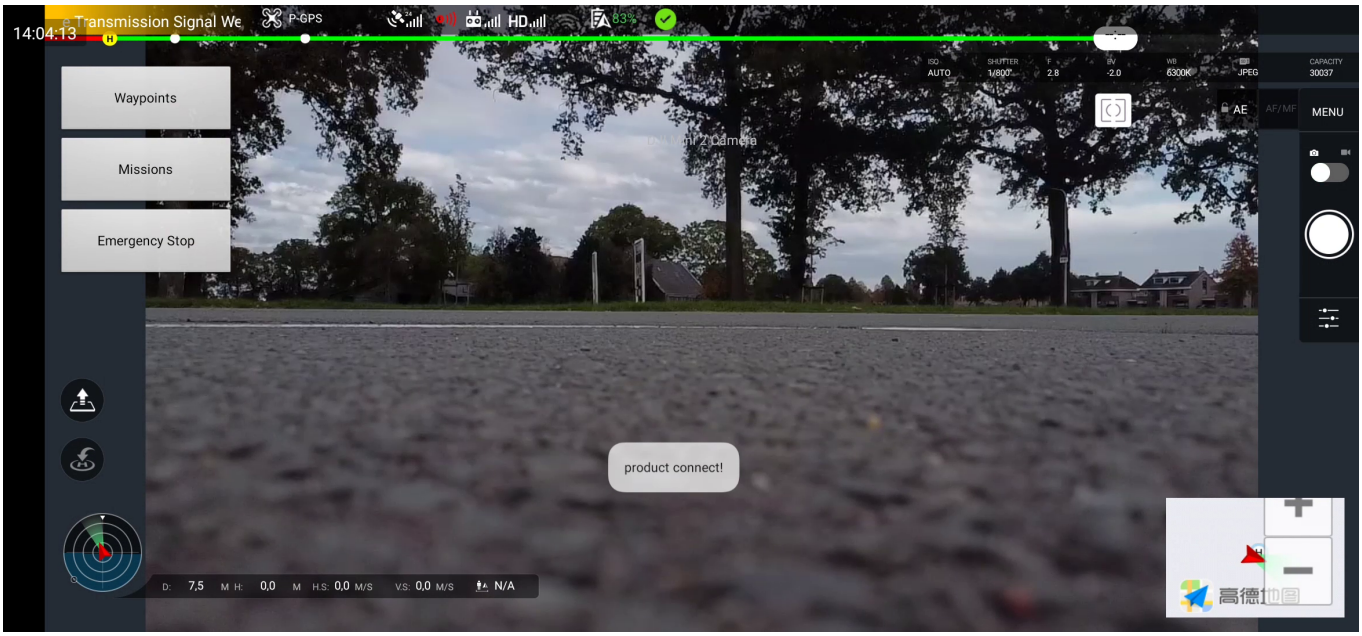

Fig. 7. The main screen of the app. The user can see a stream from the drone's camera and use the buttons to interact with missions and waypoints.

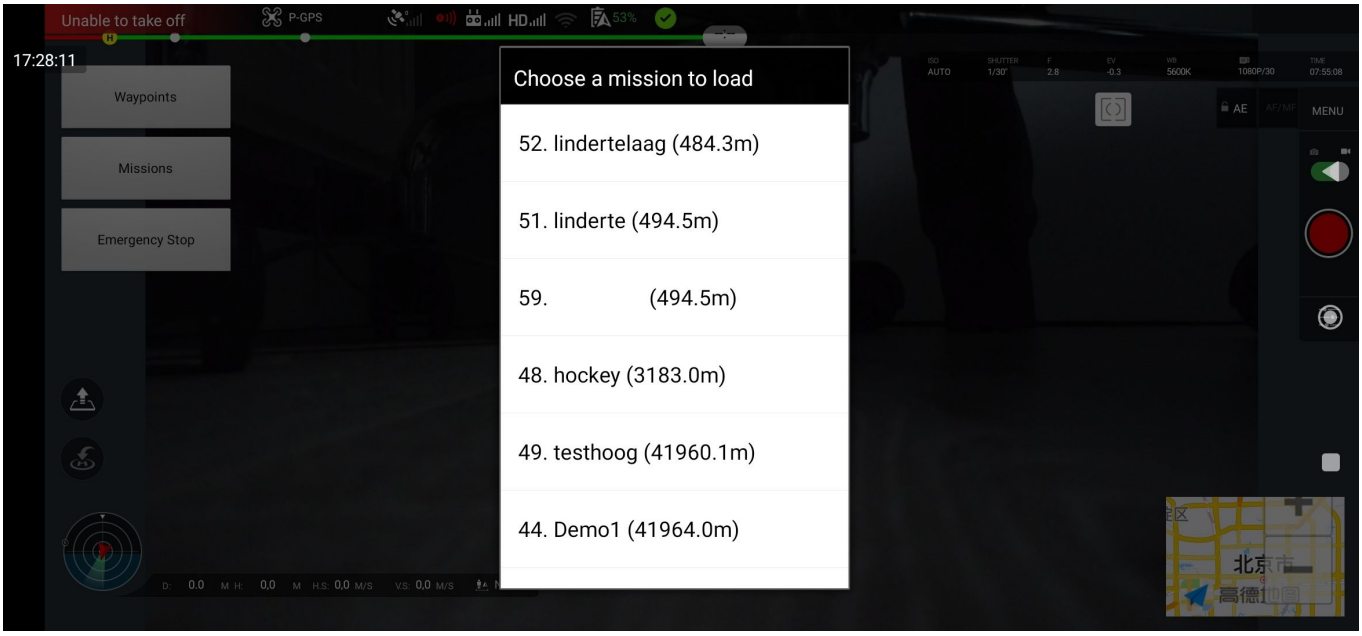

Fig. 8. This popup shows a list of missions retrieved from the API, sorted by proximity to the drone.

D DASHBOARD

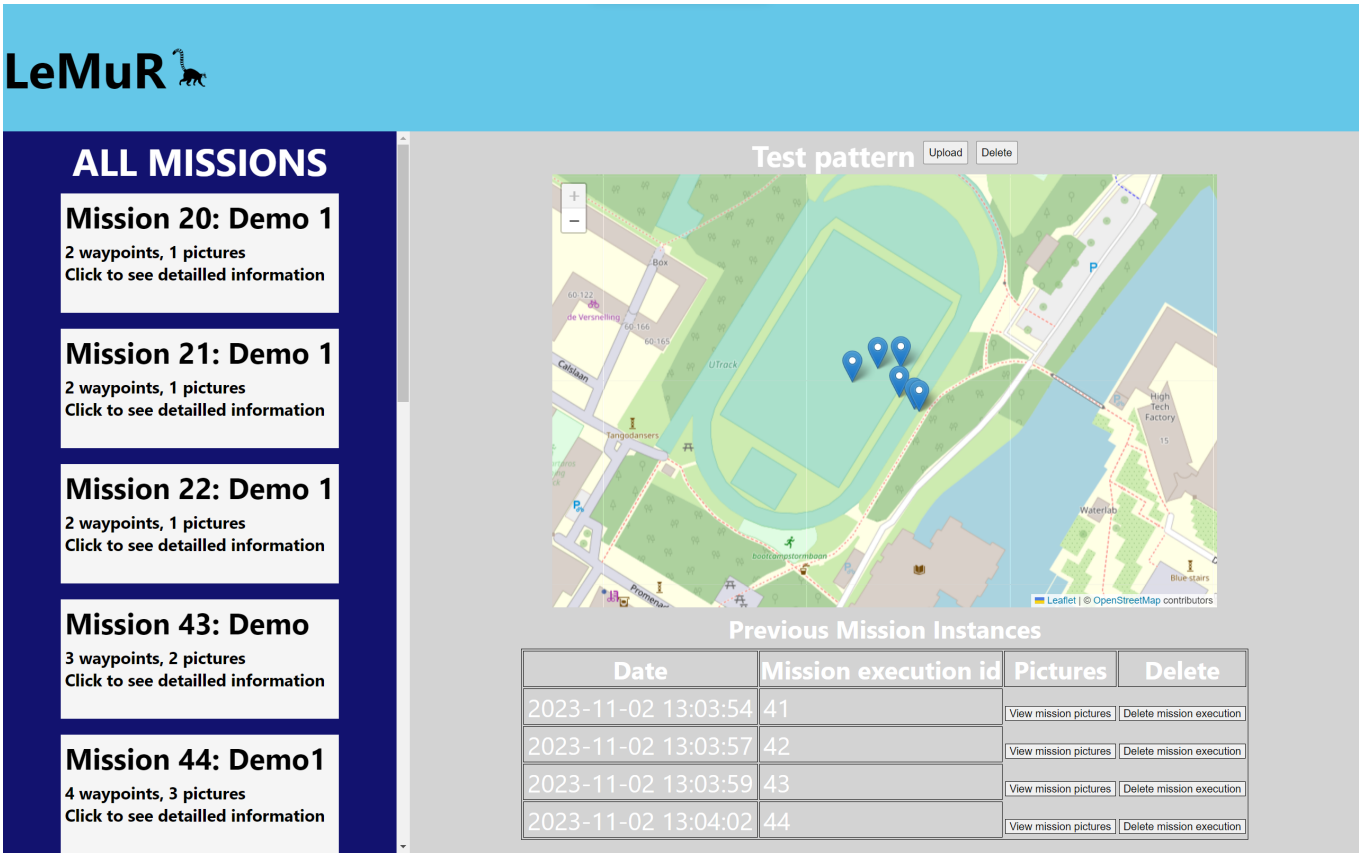

Fig. 9. Main page of the dashboard

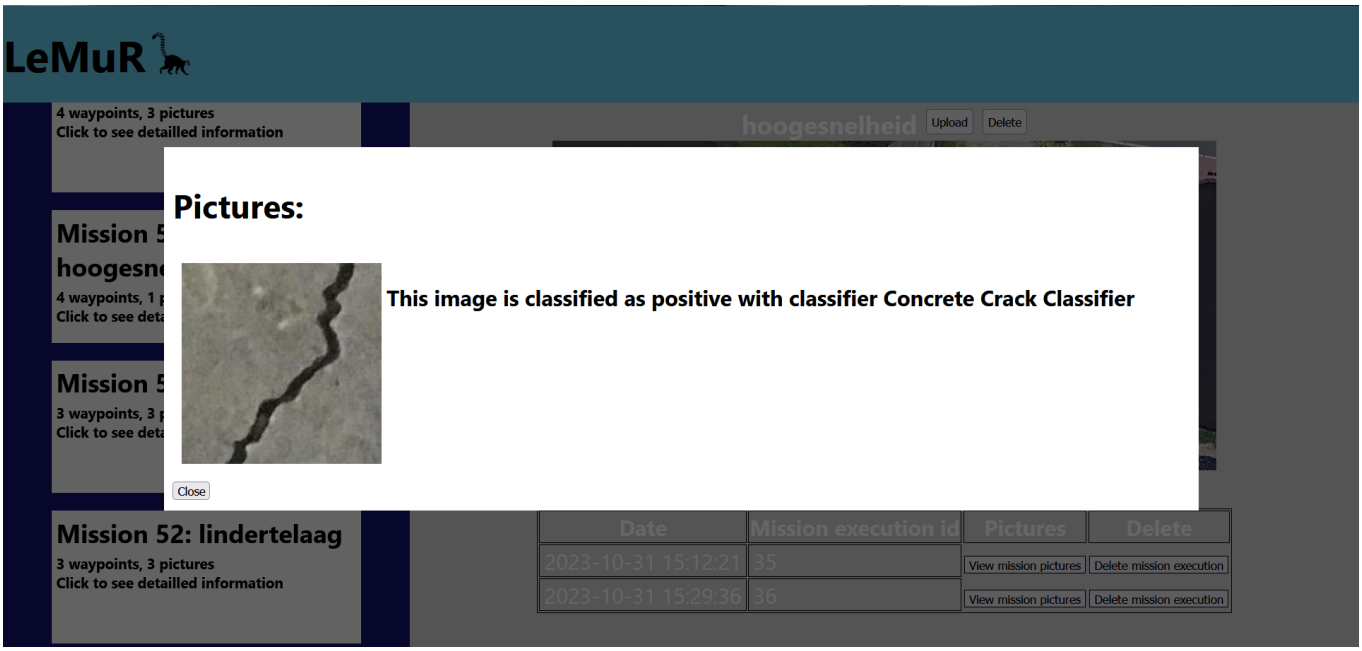

Fig. 10. Dashboard view that shows the pictures taken in a mission execution

E MACHINE LEARNING

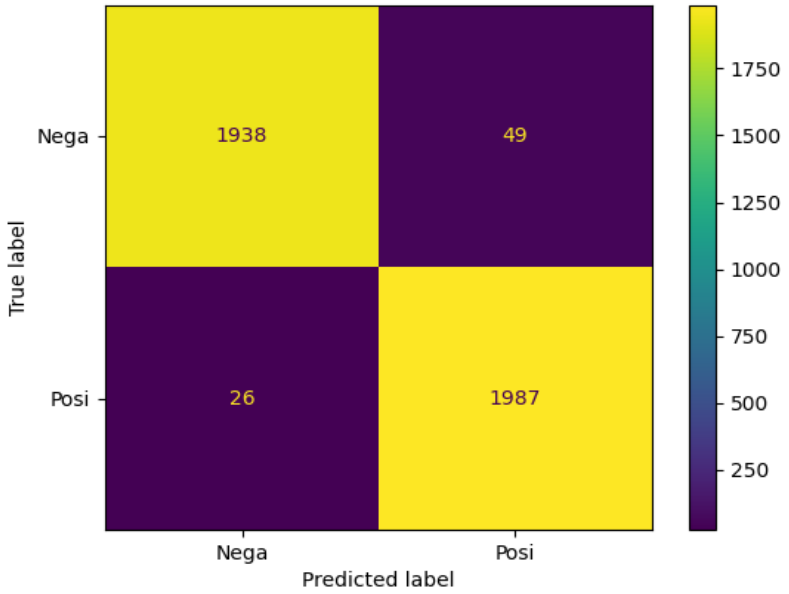

Fig. 11. Confusion matrix of predictions

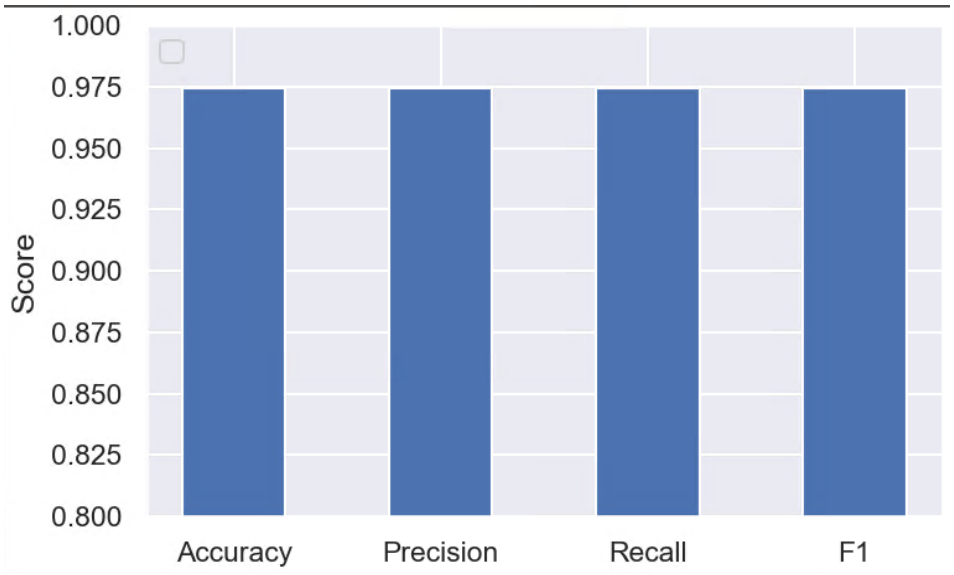

Fig. 12. Scores of classification model on test data with a 5-fold cross-validation

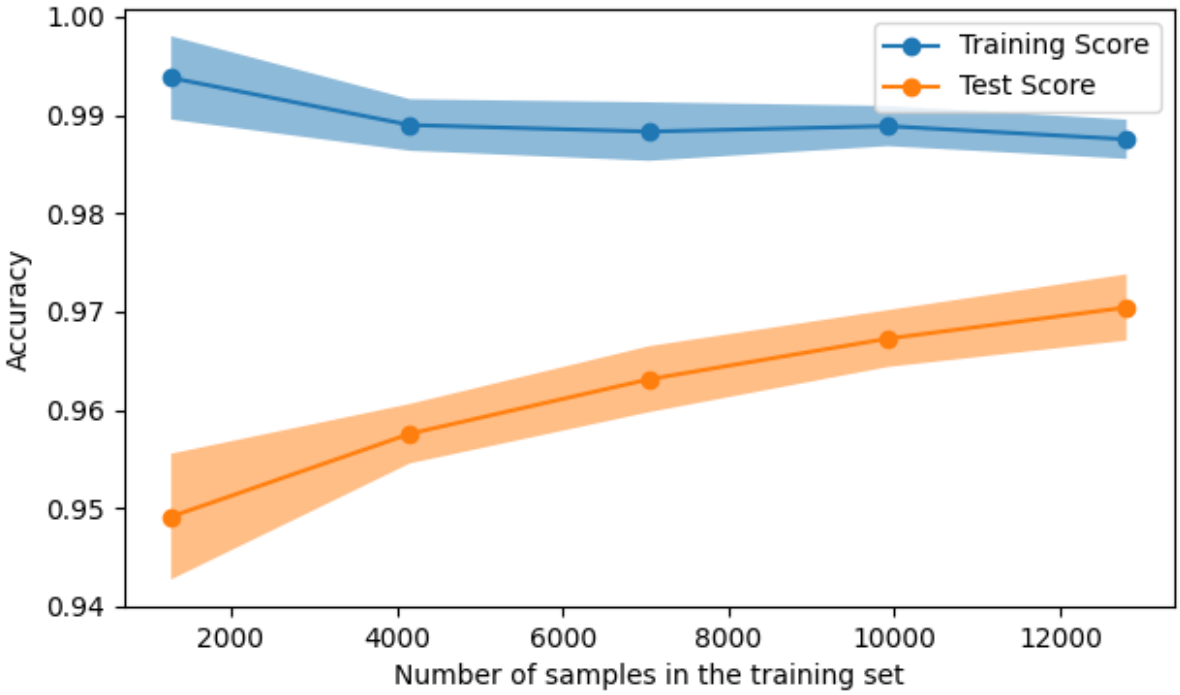

Fig. 13. Learning curve of the machine learning model