Drone Platform for Safety in Autonomous Vehicle Testing

Bachelor's thesis in Systems and Control

Mohi Eddin Bilal
Daniel Ferreira
David Mörck
Filip Sandström
Albin Svenske
Andreas Särnholm

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

BACHELOR'S THESIS 2024

Drone Platform for Safety in Autonomous Vehicle Testing

Mohi Eddin Bilal
Daniel Ferreira
David Mörck
Filip Sandström
Albin Svenske
Andreas Särnholm

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2024

Drone platform for Safety in Autonomous Vehicle Testing

Mohi Eddin Bilal
Daniel Ferreira
David Mörck
Filip Sandström
Albin Svenske
Andreas Särnholm

Supervisor: Jonas Sjöberg, Department of Electrical Engineering
Examiner: Karinne Ramirez Amaro, Department of Electrical Engineering

Bachelor thesis report 2024
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: Custom object detection model evaluated on drone footage

Typeset in LaTeX
Gothenburg, Sweden 2024

Abstract

This report details the development and implementation of an automated drone-based surveillance system for use at the AstaZero testing facility for autonomous vehicles. The system visually detects objects present at the test location and compares their position to the positions given by ATOS. If a deviation is deemed to indicate a hazardous situation, the system sends a signal to warn the test supervisor of the hazard. To bring this system into action, an Android application was developed to strategically position the drone over the test area, ensuring comprehensive monitoring and the ability to capture all test objects within its field of view. The captured footage is then transmitted to a computer for object detection, using a custom-trained YOLOv8 model. To enable the computer to communicate with ATOS in order to retrieve test specifications and send warnings, a communication application was also developed. Further specification and research into the classification of hazardous situations is required to get a system that accurately achieves the goal of an automated system.

Keywords: Object Detection, App Development, Drone, UAV, Autonomous Vehicle Testing, Surveillance, Test Safety, ROS2, ATOS, AstaZero, YOLO

Sammanfattning

Denna rapport beskriver utvecklingen och implementeringen av ett automatiskt drönarbaserat övervakningssystem för användning på AstaZeros testområde för självkörande fordon. Systemet upptäcker visuellt objekt närvarande vid testområdet och jämför deras positioner med positioner tillhandahålna av ATOS. Om en avvikelse bedöms indikera en farlig situation, skickar systemet en signal för att varna testledaren om faran. För att bygga detta system har en Android applikation utvecklats för att strategiskt positionera drönaren över testområdet, säkerställa en övergripande övervakning och förmåga att fånga alla testobjekt inom sitt synfält. Det filmade materialet sänds vidare till en dator för objektdetektering som använder en egentränad YOLOv8-modell. För att möjliggöra kommunikation mellan datorn och ATOS för att hämta testspecifika detaljer och skicka kommandon utvecklades även en kommunikationsmjukvara. Vidare forskning och specifiering av farliga situationer krävs för att uppnå målet med ett träffsäkert autonomnt system.

Acknowledgements

We would like to express our profound gratitude to everyone who contributed to this project. Foremost, we extend our thanks to Robert Brenick from AstaZero, who guided our project from start to finish and answered all of our questions promptly. We would also like to extend our thanks to to our supervisors, Jonas Sjöberg and Mikael Enelund, for facilitating our collaboration with the PSU team and engaging in discussions about various aspects of our work and feedback on our efforts.

Lastly, special thanks go to our colleagues at Pennsylvania State University — Elliott Salvatori, Tyler Vittitow, Muhammad Hussain Chaudhry, and Wenhao Xu. They not only fulfilled their responsibilities but were also readily available for urgent meetings and consistently provided the necessary assistance. Our collaboration allowed us to grow as an international team and offered us a rewarding experience of working across distances.

Mohi Eddin Bilal, Daniel Ferreira, David Mörck
Filip Sandström, Albin Svenske, Andreas Särnholm
Gothenburg, May 2024

Preface

This report covers a bachelor thesis at Chalmers University of Technology within the Department of Electrical Engineering, Division of Systems and Control. It was written in the spring of 2024, from January to May, by a group of six Chalmers students. Five students are studying Automation and Mechatronics and one student studying Computer Science Engineering.

The project was a collaboration between the Chalmers team and a team of four students from Pennsylvania State University in State College, Pennsylvania. Together, both groups exchanged ideas, feedback, material files, footage, and code. Bi-weekly meetings were consistently held, with additional sessions organized prior to project deadlines to ensure continuous progress and alignment.

The source code developed for this project is available in our GitHub repository at `https://www.github.com/filipSand/az-drone-safety-platform`.

Abbreviations

The following abbreviations are used throughout the report:

AI	Artificial Intelligence
API	Application Programming Interface
ATOS	Autonomous Vehicle Test Operating System
DJI	Da Jing Innovations
FOV	Field Of View
GUI	Graphical User Interface
IDE	Integrated Development Environment
PSU	Pennsylvania State University
SDK	Software Development Kit
RISE	Research Institutes of Sweden
ROS2	Robot Operating System 2
YOLO	You Only Look Once

Contents

List of Figures

Chapter 1

Introduction

Advanced automated driving, driver-assistance, and active safety systems for road vehicles are fields in which development and innovation are advancing rapidly. This paves the way for new combinations of safety systems, making road traffic more automated and safer both inside and around vehicles [1]. The automation of vehicles is highly dependent on active safety systems that process data from the vehicles themselves as well as interpret their surroundings by utilizing sensors [2]. Developing these sensors and systems requires developers to be in a safe environment where various settings and traffic situations can be reconstructed without posing risks to traffic and civilians. Since non-finished systems carry the potential for accidents, scenario reconstruction is often conducted in controlled tests at test tracks. The aim of this study is to increase the safety of these tests by providing an active monitoring system. AstaZero provide a test track where such tests can be conducted in a controlled environment.

In the scope of this project, a drone surveillance platform was developed to enhance safety in autonomous vehicle testing through the automatic detection of hazardous situations. The system is designed to detect unauthorized personnel and vehicles in close proximity to the test area and evaluate whether the test is being executed according to test specifications.

Currently, these tests are supervised by the test drivers themselves, who are often inside the vehicle conducting the test. This requires them to simultaneously monitor the test site, the vehicle, and the test's progress all at the same time. By utilizing the drone platform, the workload of the test driver can be reduced. The drone processes real-time information from its camera and alerts the test driver if a hazard is detected, such as the presence of more or fewer objects or pedestrians inside the test area.

1.1 Project background

This report is part of a collaboration between students at Chalmers and Pennsylvania State University, PSU. The PSU students developed the object detection model used in this project. The model is trained on data gathered from both teams, however, the training and development of the object detection model are solely the contributions of the PSU team. The computer vision model's development and results are included in the report because the final product, presented alongside this report, integrates the model as a key component. As a result, some information about the model may not be as detailed as that for other functionalities.

1.2 About AstaZero

AstaZero is a corporation owned by the Research Institutes of Sweden (RISE). Since 2012, they have operated and developed the world's first full-scale independent test environment facility outside of Gothenburg for testing, verifying, and certifying advanced automation and safety systems in road vehicles [3]. The test site consists of multiple kinds of roads and road networks [4] which result in a unique environment that supports the creation of any traffic scenario to test or verify vehicles' active safety solution [5].

1.3 Contributions

The objective of this project was to develop a software solution that works by utilizing aerial surveillance to supervise automated vehicle tests and detect possibly hazardous situations. A list of all the initial goals or requirements and desires set upon the product can be found in appendix A.

The result of the developed system is two software applications: an Android application and a Python computer program. The Android application can control the drone and allows it to fly autonomously to predetermined coordinates. The app then receives and displays the video feed from the drone's camera. The Python software can receive video from the Android application, connect to AstaZero's software controlling the autonomous vehicles, and run real-time object detection to accurately detect vehicles and pedestrians.

1.4 Demarcations

As this is a bachelor thesis project the outline and goal were predetermined before it was initiated by the group that carried out said project. This outline and goal together with other limiting factors such as time and finance excluded the group from a set of potential solutions and tools which could be interesting to explore in similar projects. Below follows some limitations that have had an effect on the choices and results throughout the project, and the motivation behind these decisions.

- The utilization of a drone to perform the test area monitoring was a requirement set by AstaZero. As such, there is no investigation of other methods to monitor the test track such as with fixed cameras or cameras and sensors mounted on the test vehicles.
- There was no hardware development during the project, only software was developed. This means that the object detection software was developed to work on a live video feed from a drone, specifically a DJI Mavic 2 Enterprise (see Section 2.2.4).
- The functionality of the object detection software was limited to recognizing objects and vehicles relevant to the AstaZero test site such as common vehicles, and pedestrians as the use of the software at other locations is not relevant.
- In this study, only a static drone is considered. This is due to time constraints in the project which did not allow the exploration of a moving drone. This also limits the types of test that can be monitored to those that fit inside the view of a static drone's camera.
- The system can not operate in poor weather, such as rain or snow. This is due to limitations of the DJI drone used in development.
- This project was limited to two visits to the AstaZero test track. This is because of the test site being fully operational and has many customers daily.

Chapter 2

Technical Background

This chapter is divided into two sections, "Related work" and "Hardware and software". Related work contains a compilation of results gathered from previous projects in the same field of science. These results have been a basis for the choices made throughout the project and also offered tools to evaluate the produced result. In the "Hardware and software" chapter, all of the hardware and software used to accomplish the end result are presented.

2.1 Related work

In this section, some related articles and conference papers are presented that builds a foundation for the computer vision that was used in the project. Understanding the concept of computer vision and its applications is crucial, as it is a central component of this system.

2.1.1 Object detection

Object detection is a computer vision task that deals with detecting visual objects of a certain class, for example, cars, humans, or dogs, in digital images. The goal of object detection is to develop models and techniques that provide the knowledge of the class and position in the image of any detected objects. In the past two decades the progress of object detection has gone through two periods: the "traditional object detection period" (before 2014) and the "deep learning-based detection period" (after 2014) [6].

You Only Look Once or YOLO for short was introduced in 2016 as one of the new deep learning-based AI architectures. Before YOLO, some approaches repurposed classifiers to perform detection. These systems took a classifier for an object and evaluated it at various locations and scales in a test image [7]. Other approaches used R-CNN [8] to first generate potential bounding boxes in an image and then run a classifier on these boxes. After the classification is done, post-processing is used to refine the bounding boxes, eliminate duplicates, and rescore the boxes based on other objects in the scene, and these pipelines are complex and slow. The creators of YOLO reframed the problem of object detection as a single regression problem that went straight from image pixels to bounding boxes and class probabilities. YOLO uses a single convolutional network to simultaneously predict multiple bounding boxes and their respective class probabilities [7].

YOLO has a mean Average Precision (mAP) of 63.4% on the Pascal VOC dataset [9] at 45 frames per second and YOLO was at the time of the release the fastest object detector on Pascal. With a mAP of 52.7% on Pascal, it was more than twice as accurate as other real-time object detectors, also, it was faster with detections at 150 Frames per second [7]. In the latest YOLO-version, YOLOv8, the model gets a mAP of 86.54% on the same dataset [10]. Because of its high product accuracy and frame rate, a small version of YOLO is often used for object detection in edge devices [11], [12], [13], [14].

2.1.2 Performance metrics

The two most significant metrics for object detection performance are accuracy and speed [6]. Speed refers to the computational effectiveness of the object detection algorithm, and accuracy refers to how well it performs. The most basic performance metrics in object detection are True Positive (TP), False Positive (FP), and False

Negative (FN), where TP means a correct detection of a ground-truth object, FP means incorrect detection of an object or misplaced detection of an object and FN means undetected ground-truth object. A common way to establish what counts as a correct detection is to use Intersection Over Union (IOU). The IOU is the overlap of the predicted bounding box B_P and the ground-truth bounding box B_{gt} divided by the area of union between them as seen in equation 2.1.

$$IOU = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})} \tag{2.1}$$

By comparing the IOU with a confidence threshold, we can decide on what counts as a correct detection. The detection will be deemed correct if $IOU \geq t$ and incorrect if $IOU < t$.
The TP, FP, and FN values can then be used to calculate precision (P) and recall (R) which are two concepts that are often used when assessing object detection. They are calculated as seen in equation 2.2 and equation 2.3.

$$P = \frac{TP}{TP + FP} = \frac{TP}{DT} \tag{2.2}$$

$$R = \frac{TP}{TP + FN} = \frac{TP}{GT} \tag{2.3}$$

Where GT is the number of ground truths and DT is the number of detections.
The precision is the ability to only identify relevant objects and recall is the ability to find all relevant cases. A low precision means that there were a lot of objects detected but only a small part of them is correct, a score of 1 means that all of the found objects are 100% correct and at the correct place. A low recall means that the model found a small part of all of the actual objects and a recall of 1 means that all of the actual objects were identified and in the correct place. The precision/recall curve is a description of how different values of the confidence threshold for the IOU affect the precision and recall. It is often a trade-off between precision and recall. A high confidence threshold will in general yield a high precision but a low recall and a low confidence threshold will yield a low precision and a high recall. To measure the performance of a model we can use the area under the precision/recall curve since we want a combination of a high precision and high recall. A large area under the curve tends to indicate high precision and high recall [15].

The area under the curve can be calculated in two ways. The N-point interpolation method or the all-point interpolation method. For the all point method equation 2.4 is used.

$$AP_{all} = \frac{1}{GT} \sum_{R \in R_{all}} P_{max}(R) \tag{2.4}$$

where $P_{max}(R) = \max_{R':R' \geq R} P(R')$. The recall is discrete in the range [0,1] with an interval of $\frac{1}{GT}$, therefore $R_{all} = \{\frac{1}{GT}, \frac{2}{GT}, ..., \frac{TP}{GT}\}$. $P(R')$ is the precision at the interpolation point R'. $R' \geq R$ means that $P_{max}(R)$ is the maximum precision among the interpolation point R and subsequent interpolation points [16]. The N-point interpolation is very similar but uses N points. For example, AP_{11} uses 11 points and the equation can be seen in equation 2.5.

$$AP_{11} = \frac{1}{11} \sum_{R \in R_{11}} P_{max}(R) \tag{2.5}$$

Where $R_{11} = \{0, \frac{1}{10}, \frac{2}{10}, ..., 1\}$. The mean Average Precision (mAP) is used to measure the accuracy of object detectors over all classes in a database [15]. This is calculated by taking the average AP over all classes, as can be seen in equation 2.6.

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{2.6}$$

The F1 Score is a combination of the recall and the precision scores at different threshold values. This score is used to assess the overall performance of the model in regards to both P and R. It is calculated by taking their harmonic mean, as can be seen in equation 2.7.

$$F1 = \frac{1}{\frac{1}{P} + \frac{1}{R}} \tag{2.7}$$

2.2 Hardware and software

During this project, several already existing tools were used in the process of design and development. In this section, the tools and software kits used in the project are presented. The information presented is an overview of the different tools and is not a description of how they were used in the project. These tools will be referenced further on in the report.

2.2.1 ROS

Robotics Operating System or ROS, is a set of software frameworks for developing software for robotics systems [17]. It provides services such as hardware abstraction, low-level device control, message passing, and package management. The types of communication that are used in ROS are synchronous communication over *services*, asynchronous streaming of data over *topics*, and data storage in a *Parameter server*.

ROS2 is the new version of ROS which is continuously getting updated. ROS2 was made to support real time programming, which was not a part of the first version of ROS [18]. It was also made to support a wider variety of computing environments and more modern technology.

2.2.2 ATOS

Autonomous Vehicle Test Operating System (ATOS), is a software control center and coordinator for scenario-based testing of autonomous vehicles in real-time developed by AstaZero [19]. ATOS is based on ROS2 and uses ROS2 topics for data that is published continually, for example, the coordinates of the test objects, and ROS2 services for things that are more constant, like the coordinates of the test origin and the test states. The test states in ATOS are the states that control the test objects, a basic chart over the most common test states is presented in Figure 2.1. ATOS reads trajectories and controls self-driving vehicles participating in the test by executing scenarios specified in test files. It uses GPS to assure precise and repeatable executions.

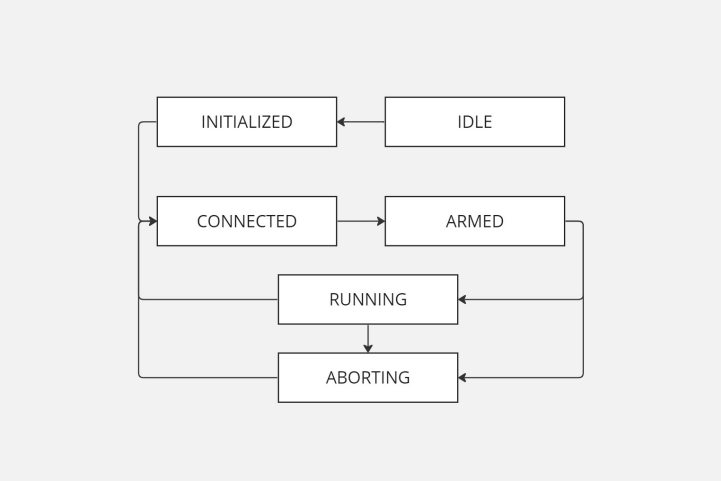


Figure 2.1: The ATOS test object states

ATOS is usually run together with Foxglove Studio commonly used as an interface for controlling the testing rather than directly publishing and subscribing to ROS2-topics. From this page, the user can initialize, arm, start, and abort an ATOS test as well as visualize where the test objects are currently positioned. The Foxglove control page can be seen in Figure 2.2. In this image, the test has been initialized and the lines that are shown on the lower middle and right window is the test object trajectories.

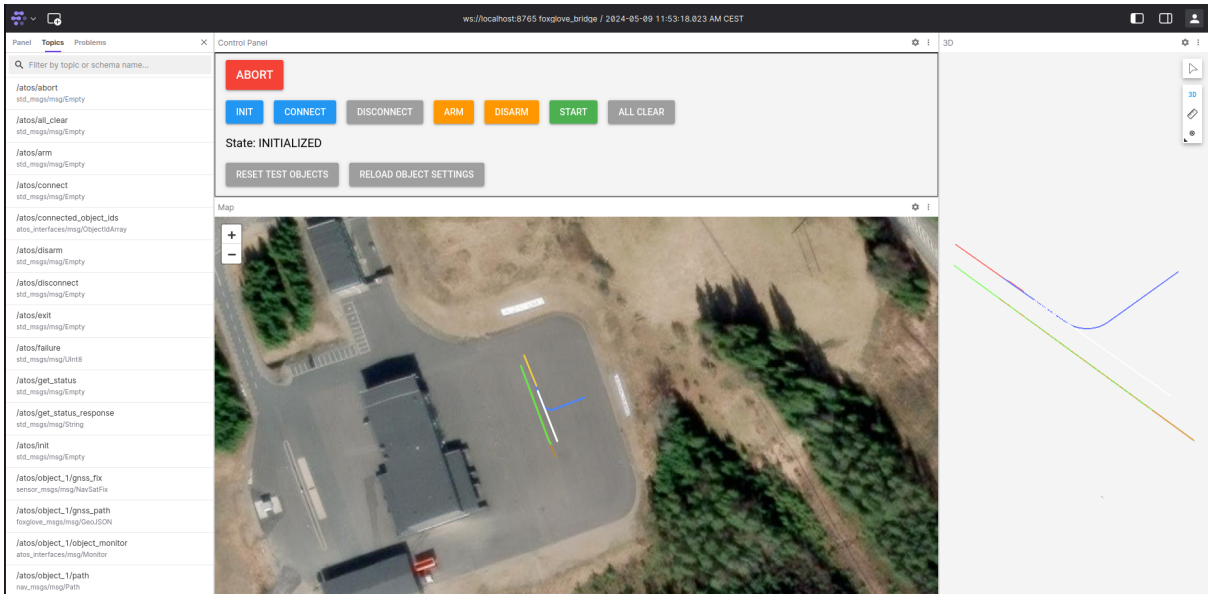


Figure 2.2: The Foxglove interface

2.2.3 Android Studio and Android application development

There are several ways to develop Android applications. Google, the maintainer of Android, has created its own integrated development environment, IDE, to develop native Android applications called Android Studio [20]. Native Android applications are written in the Java programming language. An advantage of using Android Studio is that there are many third-party Software Development Kits (SDKs) that speed up the development process and abstract low-level implementation details. Both the WebSockets communications package used in the application and the DJI API are examples of such third-party SDKs.

2.2.4 Hardware

The drone used in this project was a DJI Mavic 2 Enterprise [21], as pictured in Figure 2.3. The drone comes with a remote control that can be used to fly it manually, but the drone can also fly automatic pre-programmed missions to user-specified waypoints and perform custom actions on these. These pre-programmed operations are in this report referred to as waypoint missions. The drone has an onboard camera mounted on a gimbal. It is possible to connect the controller to a mobile phone via a USB cable. Through this connection, it is possible to retrieve the output from the drone's built-in camera as well as send commands to the drone, such as the aforementioned waypoint missions.

Figure 2.3: The DJI Mavic 2 Enterprise Drone with controller

The PSU team also had access to a drone, a DJI Mini 2 [22]. This drone could also record video and fly pre-programmed missions but is only partially compatible with the DJI API discussed in Section 2.2.8.

2.2.5 WebSockets

WebSockets is a communications protocol that enables bidirectional communication [23]. This two-way communication enables both server and client to send requests to each other. In use cases where both server and client want to initialize communication, this can make for lower overhead and reduced latency. Rather than having the client regularly polling the server with requests to stay up to date on changes such as with unidirectional communication, the server can send a request to the client when changes have occurred. Figure 2.4 illustrates the difference between communication using HTTP and using WebSockets. Another advantage of WebSockets is that it provides a high-level interface which makes it easy to implement data communication.

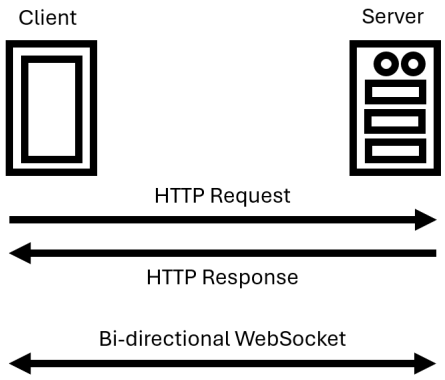


Figure 2.4: Comparison between communications with HTTP and WebSockets.

2.2.6 Sockets

A socket, also known as a BSD a POSIX socket, or a network socket when implemented over a network, is a software mechanism for interprocess communication (IPC) [24] It establishes a communication channel and provides endpoints for data to two separate processes, which can be written to and read from. Sockets are low-level and generic and do not provide any higher-level functionality apart from direct data exchange. Widely adapted by modern operating systems, sockets are the most used form of IPC today [25] and are used to implement many communication protocols.

2.2.7 WebRTC

Short for Web Real-Time Communications, WebRTC is an open-source project designed to facilitate real-time, text, video, and audio communication capabilities between web browsers and devices [26]. Using JavaScript APIs, WebRTC can establish peer-to-peer communications across the Internet and mobile applications without requiring additional plugins or software installations for text, audio, or video interactions. Since data transfer occurs in real-time with WebRTC, engaging in audio and video communication directly from a webpage without custom interfaces is possible [27].

2.2.8 DJI API

The DJI Application programming interface or API [28] is an API delivered by the drone manufacturer DJI. The API works similarly to any other API with the purpose of giving a user control of functionalities without being limited to existing applications and services. The API is delivered within a SDK designed for Android mobile devices which is mentioned throughout the project.

The DJI API gives the user the opportunity to use specific abilities of the drone to accommodate their own personal needs. The functionalities include functions and classes for drone control, states, and control for different accessories compatible with the drone. DJI offers a collection of guides for implementing and using the API in different areas which includes camera stream, waypointmissions, GUI:s, and a few more.

Chapter 3

Method

Below follows the methodology of development to reach the objective described in the introduction, see 1, firstly for the system as a whole and thereafter for the separate sub-systems.

3.1 System overview

To achieve the objective the complete system was divided into four smaller sub-problems, illustrated as puzzle pieces in Figure 3.1. The sub-problems were: ATOS communication, Android application, video streaming, and object detection. These four parts form a comprehensive system, represented in Figure 3.2, where colored boxes denote sub-systems and the gray boxes highlight significant components of each sub-sysetm.

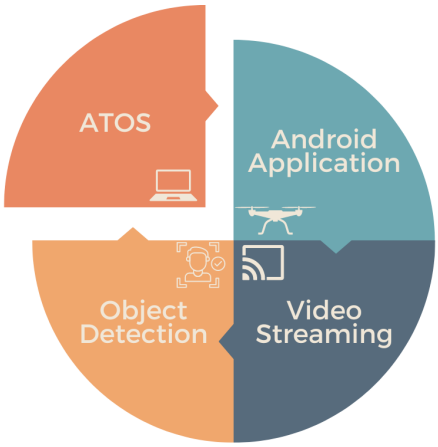


Figure 3.1: An overall system image

- **ATOS communication:** The system as a whole revolves around supervising a test run by ATOS, see Section 2.2.2. Extracting information from here is integral to supervising and giving correct responses regarding the test scenario. The coordinates of the different test objects need to be extracted from ATOS. These coordinates are then integrated into the drone via the Android application as well as compared with the result produced by the object detection software. ATOS is also responsible for controlling the self-driving vehicles in the test field while running a test scenario.
- **Android application:** The Android application serves as the control interface for the drone and a hub for the drone component in the system. It manages parameters responsible for height, velocity, and camera gimbaling as well as executes built-in commands in the drone, see Section 2.2.4. Using the test objects' coordinates extracted from ATOS, an algorithm have been developed to guide the drone to the center of the test objects and calculate the optimal height required to capture all the test objects within its field of view (FOV).
- **Video Streaming:** To apply the object detection algorithm, the live camera feed of the drone is transmit-

ted to a computer using a live server. Since the drone is connected to the phone via its controller, the camera feed is transmitted to the computer via the phone. This connection facilitates real-time transmission of the drone's video feed to the computer for processing.

- **Object Detection:** Object detection involves developing an algorithm trained on data related to various test objects, such as vehicles, and pedestrians. This part of the system is crucial for real-time observing the test environment and detecting potential hazardous situations by recognizing people and vehicles as well as their positions.

The integration of these four components forms a complete system capable of conducting autonomous vehicle tests effectively that prioritize safety. While interconnected, each part can be developed and optimized independently to a certain limit until incorporated with each other to form a bigger system running in parallel with an ATOS test scenario.

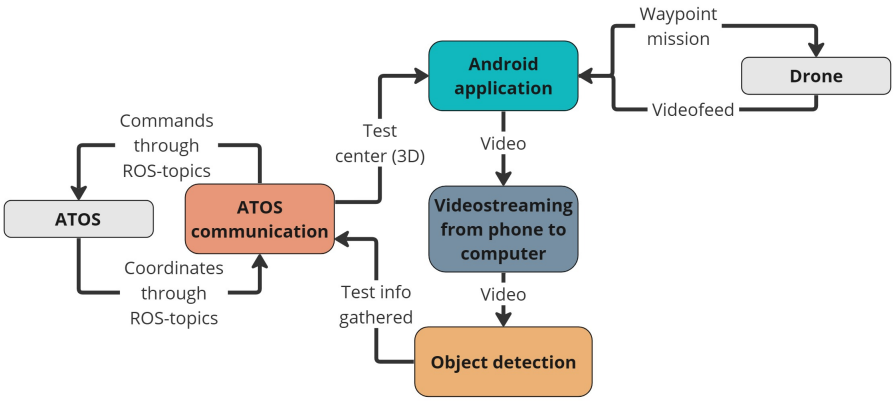


Figure 3.2: System goal divided according to methodology

3.2 ATOS and communication software

In order to compare the actual coordinates of test objects to the ones specified in the test, a solution for fetching the test coordinates was required. ATOS uses ROS2 2.2.1 for communication, this means that instead of trying to communicate with some module in ATOS, information can instead be retrieved straight from the ROS2 topics and services. To automate the Python API for ROS2, rclpy [29] is used to communicate with ROS2 in Python. Rclpy is the Python client library for ROS2, enabling Python programs to communicate with ROS2 topics, services, and actions. This is used in what is referred to as the communication software.

Since ATOS uses ROS2 for communication, it is important to find out what topics and services ATOS uses. This way, the topics can be subscribed and published to and the services can be called. This means that the communcation software can act like it is a part of ATOS. To find out to what ROS2 topics ATOS publishes, the command line tool ROS2 was used. By starting ATOS and entering the command `ros2 topic list -t`, all of the topics and their types is printed. This can be seen in Figure 3.3a. This was used in combination with `rostopic echo <topic_name>`, which is a command that subscribes and prints all of the messages published to that topic[30]. By doing this for several of the topics, the correct information sought could be found, and also the type of message that it was sent with. The type of message is needed to be able to subscribe to topics in code.

A similar approach was taken for finding the relevant services. The command `ros2 service list -t` was used to list all services and their types. To find out how the interface for the type looks like, the command `ros2 interface show <type_name>` can be used. The interface is what input data is expected to be sent when calling the service and the output that is returned from the service. In Figure 3.3b the interface for getting the trajectories for the objects from ATOS is shown.

3.2.1 Getting coordinates and trajectories from ATOS

The GPS coordinates of each test object are published almost 100 times per second, this was realised by using `ros2 topic hz atos/object_<object id>/gnss_fix`. To listen to the coordinates being published, the topic *gnss_fix* for each object has to be subscribed to. The full topics look like this: *atos/object_<object id>/gnss_fix*. This requires the object IDs, so before the coordinates can be subscribed to, the IDs of all objects have to be found. To get the object IDs, the service *GetObjectIds* has to be called. This service returns a list of all of the

```
davidmorck@Ubuntu3:~$ ros2 topic list -t
/atos/abort [std_msgs/msg/Empty]
/atos/all_clear [std_msgs/msg/Empty]
/atos/arm [std_msgs/msg/Empty]
/atos/connect [std_msgs/msg/Empty]
/atos/connected_object_ids [atos_interfaces/msg/ObjectIdArray]
/atos/disarm [std_msgs/msg/Empty]
/atos/disconnect [std_msgs/msg/Empty]
/atos/exit [std_msgs/msg/Empty]
/atos/failure [std_msgs/msg/UInt8]
/atos/get_status [std_msgs/msg/Empty]
/atos/get_status_response [std_msgs/msg/String]
/atos/init [std_msgs/msg/Empty]
/atos/object_1/gnss_fix [sensor_msgs/msg/NavSatFix]
/atos/object_1/gnss_path [foxglove_msgs/msg/GeoJSON]
/atos/object_1/object_monitor [atos_interfaces/msg/Monitor]
/atos/object_1/path [nav_msgs/msg/Path]
/atos/object_2/gnss_path [foxglove_msgs/msg/GeoJSON]
/atos/object_2/path [nav_msgs/msg/Path]
/atos/object_3/gnss_path [foxglove_msgs/msg/GeoJSON]
/atos/object_3/path [nav_msgs/msg/Path]
/atos/object_4/gnss_path [foxglove_msgs/msg/GeoJSON]
/atos/object_4/path [nav_msgs/msg/Path]
/atos/object_5/gnss_path [foxglove_msgs/msg/GeoJSON]
/atos/object_5/path [nav_msgs/msg/Path]
/atos/object_6/gnss_path [foxglove_msgs/msg/GeoJSON]
/atos/object_6/path [nav_msgs/msg/Path]
/atos/object_state_change [atos_interfaces/msg/ObjectStateChange]
/atos/objects_connected [std_msgs/msg/Empty]
/atos/reload_object_settings [std_msgs/msg/Empty]
/atos/remote_control_disable [std_msgs/msg/Empty]
/atos/remote_control_enable [std_msgs/msg/Empty]
/atos/replay [std_msgs/msg/Empty]
/atos/reset_test_objects [std_msgs/msg/Empty]
/atos/start [std_msgs/msg/Empty]
/atos/start_object [atos_interfaces/msg/ObjectTriggerStart]
/atos/state_change [atos_interfaces/msg/StateChange]
/atos/stop [std_msgs/msg/Empty]
/atos/v2x_message [atos_interfaces/msg/V2x]
/clicked_point [geometry_msgs/msg/PointStamped]
/initialpose [geometry_msgs/msg/PoseWithCovarianceStamped]
/move_base_simple/goal [geometry_msgs/msg/PoseStamped]
/parameter_events [rcl_interfaces/msg/ParameterEvent]
/rosout [rcl_interfaces/msg/Log]
```

(a) The list that shows up after entering `ros2 topic list -t`

```
davidmorck@Ubuntu3:~$ ros2 interface show atos_interfaces/srv/GetObjectTrajectory
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.

uint32 id
---
bool success true
uint32 id
CartesianTrajectory trajectory
        std_msgs/Header header
                builtin_interfaces/Time stamp
                        int32 sec
                        uint32 nanosec
                string frame_id
        CartesianTrajectoryPoint[] points
                builtin_interfaces/Duration time_from_start
                        int32 sec
                        uint32 nanosec
                geometry_msgs/Pose pose
                        Point position
                                float64 x
                                float64 y
                                float64 z
                        Quaternion orientation
                                float64 x 0
                                float64 y 0
                                float64 z 0
                                float64 w 1
                geometry_msgs/Twist twist
                        Vector3  linear
                                float64 x
                                float64 y
                                float64 z
                        Vector3  angular
                                float64 x
                                float64 y
                                float64 z
                geometry_msgs/Accel acceleration
                        Vector3  linear
                                float64 x
                                float64 y
                                float64 z
                        Vector3  angular
                                float64 x
                                float64 y
                                float64 z
        string controlled_frame
        string name
```

(b) What shows up after entering `ros2 interface show atos_interfaces/srv/GetObjectTrajectory`

Figure 3.3: ROS2 output for fetching topics and trajectories

object IDs. When this has been acquired, the topics for all of the different objects' coordinates can be subscribed to. The subscription means that each time a message is published, the script gets a callback that catches the message. To be able to access the coordinates from anywhere in the code, the callback function stores the new coordinate in a global variable after it has required a lock that is used for reading and writing to the variable. To see the subscription and callback functions, see appendix B.1.

To get the trajectories from ATOS, the service *GetObjectTrajectory* was used. When the interface was inspected, it was found that it requires an object ID to return that object's Cartesian trajectory. If the IDs of all objects haven't been fetched already, they will be retrieved first, and then all the trajectories of the different test objects will be obtained using the ROS2 service. Since these trajectories do not change during the test, fetching them once is sufficient. They are saved in the Python script as lists containing the coordinates (longitude, latitude, and altitude) for all points along each trajectory. The code for getting the trajectory for a test object can be seen in appendix B.2.

3.2.2 Reading and changing the object control state of ATOS

The ATOS control states are shown in Figure 2.1 and just like the coordinates and trajectories, they can be controlled and read using ROS2 services and topics. To find out how to change the ATOS state, the code behind the Foxglove GUI mentioned in Section 2.2.2 was inspected. The code is open to the public and can be viewed on RISE:s Github repo [31]. The buttons at the top of the page in Figure 2.2 are used to control the ATOS control states. By inspecting the code behind these buttons it was found out that there was a separate ROS2 topic for each type of control state change command. For example the topic for initializing the test is *atos/init*. These topics were expecting empty messages, which can be seen at the top of image 3.3a. This means that ATOS reacted to receiving a message from a certain topic and reacted differently depending on which topic it was. This was used in the Python script to control the states.

The way of getting the current test state was also found in the same script as the buttons. It is a service called *GetObjectControlState* that is called to request the current state. It returns a number between 0 and 10 which correspond to different states. Rclpy contains a way of easily creating a timer that calls a function each time a certain number of seconds has passed. This can be used to fetch the ATOS states continuously. This can be seen in the code in appendix B.3.

3.2.3 Drone positioning in 3D

Since the drone will hover at a fixed position as mentioned in Section 1.4, for the drone surveillance to work sufficiently the drone needs to be positioned in a way that enables it to cover the majority of the test with its camera. To solve this problem the following methods introduced in this chapter were used, with the first calculation calculating at which altitude the drone needs to fly at to cover a certain area. To determine this altitude, a target area to cover was needed, AstaZero was consulted with the question, and the result was a requirement of covering 500 m^2 and a desire to cover 1000 m^2, this desire was later increased to 1500 m^2.

To calculate at which height the drone needs to fly for the camera to cover a certain area, two methods were proposed. The initial method was based on [32], which utilizes internal distances within the camera to determine the area. The second method is produced within this project and should only be regarded as an estimation based on trigonometry by a novice regarding vision and optics systems. However, this method has created results more in line with the feedback gathered from the validation method that will be presented later and is thereby chosen as the method used in this project. The calculations are presented below with an associated image of the model which can be seen in Figure 3.4.

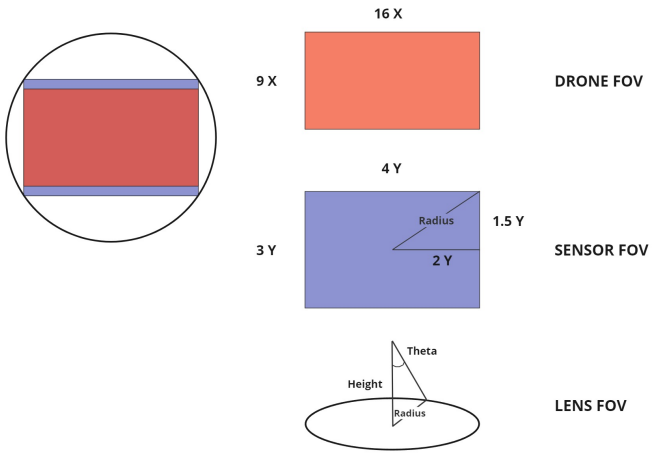


Figure 3.4: A visual model associated with the proposed height calculations.

The area to be covered by the drone is an input to the script, the input should be in the unit m^2. As can be seen in 3.1.

$$\text{A} = [\text{input from user}] \tag{3.1}$$

The field of view (FOV) of the camera is a necessity in the calculations. For the drone used in this project, the FOV is according to the DJI website 82.6° [21]. The angle of the sought triangle in "LENS FOV" in Figure 3.4 is this angle divided with two, as equation 3.2 accordingly.

$$\theta = \frac{82.6}{2}° \tag{3.2}$$

The aspect ratio tells one the relationship between the width and the length of an image. The displayed footage from the drone is of 16:9 aspect ratio and the area of a rectangle is calculated by multiplying the width by the length. With this information, the x to which the width and length need to be scaled to in order for it to cover the previously entered area can be calculated. This x is calculated with the following equation 3.3.

$$x = \sqrt{\frac{\text{A}}{16 \cdot 9}} \tag{3.3}$$

This rectangle has the same width as that of the sensor as can be seen in Figure3.4, however, the sensor has a 4:3 aspect ratio and its scalar y. This y is calculated by setting the width of the rectangles equal to each other and then dividing both sides with 4. The resulting equation is equation 3.4.

$$y = \frac{16 \cdot x}{4} \tag{3.4}$$

The radius of the circle is then calculated with Pythagoras theorem where the hypotenuse is the radius, half of the sensor width is the adjacent, and half of the sensor length is the opposite. Equation 3.5 is Pythagoras with inserted values and the variable y.

$$\text{radius} = \sqrt{((2 \cdot y)^2 + (1.5 \cdot y)^2)} \tag{3.5}$$

The radius, the height of the drone, and the diagonal distance from the drone to the point where the sensor FOV intersects with the lens FOV in the image form another triangle. Where the hypotenuse is the aforementioned diagonal distance, the opposite is the radius and the adjacent is the height. The height is then calculated with the known angle θ and the radius which returns the height in meters with the following equation 3.6.

$$\text{height} = \frac{\text{radius}}{\tan(\theta)} \tag{3.6}$$

The previous calculation is used to decide the altitude to fly at in order to cover a test of a certain size, however, the drone might not cover the test itself if it is positioned indifferently in the latitudinal and longitudinal dimensions.

In each autonomous vehicle test run at the AstaZero test site, a test origin is generated, and all of the object trajectories throughout the test are predetermined and relative to the test origin. The test origin is however not the center or origin of the test, it is an arbitrary position in close proximity to the test which makes it a rather poor positioning point for the drone. Instead, the idea behind the positioning was to position the drone in the mean position with regards to the test objects' relative locations throughout the test. That way the position of the drone will most often end up in the middle of the test and it will also be weighted to be closer to positions where more objects are and where objects are for a longer duration. As previously mentioned all of the test objects' positions are described in two dimensions relative to the test origin, so by gathering all of these positions it is possible to calculate the arithmetic mean of positions in the two dimensions. The arithmetic mean is calculated with the following equations, the mean in the x and y direction is calculated with equation 3.7 and 3.8 accordingly.

$$\bar{x} = \frac{x_1 + x_2 + x_3 + ... + x_n}{n} \tag{3.7}$$

$$\bar{y} = \frac{y_1 + y_2 + y_3 + ... + y_n}{n} \tag{3.8}$$

Where the y-axis is pointing straight north and the x-axis is pointing east. When iterating through all of the coordinates in the trajectory list, the coordinates that are located the furthest away from the other could be saved, in other words, $[min_x, y]$, $[max_x, y]$,$[x, min_y]$ and $[x, max_y]$ would be saved. By storing these values, a calculation of in what of the two directions the test objects travel the furthest can be performed, this travel can be used to calculate the test area that is used as an input into the height calculation. Because of the aforementioned aspect ratio of the footage gathered from the drone the rotation of the drone is important as well, since the aspect ratio allows for more visuals in the horizontal than the vertical direction. This makes this travel further useful. By rotating the drone so it is perpendicular to the direction in which the test objects travel the furthest the drone retrieves more useful visual data from the test.

All of these mentioned aspects and ideas are the ideas behind the positioning calculations incorporated in this project, a visual representation of the idea can be seen below in Figures 3.5,3.6,3.7. In these three figures, the blue and orange boxes resemble two vehicles at different timestamps throughout a test. The star resembles the drone and the surrounding rectangle is the drone's camera's viewable area. The viewable area of the drone is in the figures fixed in contradiction to the real system that would utilize the endpoints of x and y-travel in order to calculate the height and the resulting area. The code with which this is done can be found in appendix B.4.

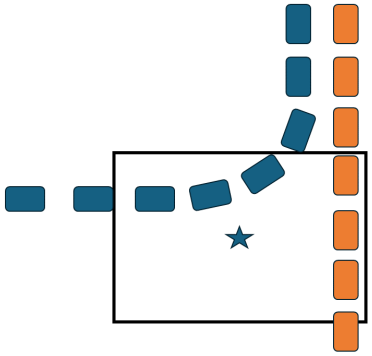


Figure 3.5: Drone position at test origin

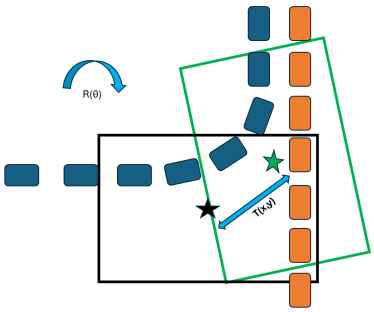


Figure 3.6: Drone translation and rotation for better position

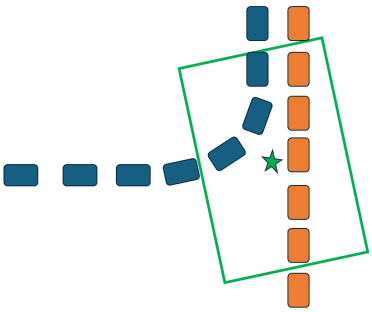


Figure 3.7: Drone positioned in a more favourable position

To validate the height, position, and orientation calculations, Google Earth has been used. Within Google Earth

a tool is available that lets the user map out a set of points with connecting lines, by encapsulating a region within these lines the area of that region is calculated. By combining this tool with footage taken by the drone at different altitudes one can gather the visible region from that altitude in the footage and portray that region into Google Earth and from there get a calculated area, an example of this can be seen in Figure 3.8.

(a) Drone footage taken from 50 m

(b) The same region entered in Google Earth

Figure 3.8: The height validation method used
Imagery ©2024 Airbus, Lantmäteriet/Metria, Maxar Technologies, Map data ©2024

With a combination of virtually replaying the test and displaying all of the test object's trajectories in Foxglove, the formula can be used and see at what coordinate and orientation the drone would want to position it self. By entering this position and rotation into Google Earth the output of the placement method can be compared with the visuals in Foxglove. This is used as a basis to verify whether or not the output is reasonable, an example of this will can be seen in Figure 4.4.

3.2.4 Recording autonomous ATOS tests

To test the communication with ATOS, several real autonomous vehicle test scenarios using ATOS, at the AstaZero test track, was recorded both using the drone camera and using the tool *ros2 bag*. ROS2 bag is a tool that records all the ROS2 messages being passed in the different topics. This can later be played back to "relive" the situation from the perspective of ROS2 [33]. This was important from a development perspective since it was not possible to be at the test track at all times.

To record a test drive, the drone was flown up to about 30 metres and the camera recording was started [33], [34]. To record all ROS2 topics avaliable, the command `ros2 bag record -a` was used in a terminal. The ATOS test was then initialized by pressing the buttons in Foxglove in the following order while being connected to an actual car: *init*, *connect*, *arm*, and then *start*. After the test drive was finished, the camera recording was stopped and the ROS2 topic recording was stopped by pressing `Ctrl + C` in the same terminal as the recording was running in. The recording is then automatically time stamped and saved in a folder on the computer.

To play back a recordning, the command `ros2 bag play <name of folder>` was used. This was used to "pretend" that an actual test drive was being conducted.

3.3 Android application

The responsibility of the Android application is to control a drone, taking coordinates as input and sending out a live video stream which can be processed by a computer doing object detection and ATOS communication. A design goal for the application is to minimize the amount of interaction the user has to do with the phone. Instead, interactions and functionalities used during an ATOS test scenario are to be centralized to the computer running the object detection. The functionalities needed in the application are as follows:

- **Videostreaming from drone to phone:** A camera stream from the drone needs to be fetched in order to be forwarded to the computer.
- **Dronecontrol:** To make sure the drone automatically flies to the calculated and desired coordinates.

The first step was to build the application without integrated support for communication with ATOS. This entails building a simple drone control application with functionalities suited to the objective of the project. To achieve this, guides from DJI regarding their Android SDK were used [35]. Some modifications had to be done to the code in order to adapt the functionality to better fulfill the demands of the system.

When the drone control application was operational, a bridge between the application and a computer would have to be designed so that the application could receive instructions from the computer and so that the video feed captured by the drone could be streamed back to the computer.

The general method of application development has been to divide the application into sub-functionalities. From there, the process was to build them separately, test them in isolation, test them together with the whole system, and thereafter build the next sub-functionality.

3.3.1 Video streaming from drone to phone

The DJI SDK [28] provides a way to capture the camera feed from the drone and display it in a custom application. The implementation is built on a tutorial [36] which is provided by DJI. As such, the code was rewritten to work on newer versions of Android. The SDK provides methods to both display the video feed and to capture the raw video data to process it for further use. This functionality would have been used to transmit the stream to the computer, had that functionality been successfully implemented.

3.3.2 Measurement of glass-to-glass delay

The glass-to-glass delay is defined as the time between an object being visible in a camera to the time it is visible on a user's screen [37]. This test measures the delay of the system as a whole and includes all processing and transmission delays. The test was performed by having the drone's camera, a laptop screen, and the Android phone displaying the camera feed all in frame of an iPhone 11 Pro recording in high speed mode. This mode has a frame rate of 240 FPS [38]. The frame rate was also verified by the frame rate of the video file. The setup is shown in Figure 3.9 where both the phone recording the test and the drone are both viewing the laptop.

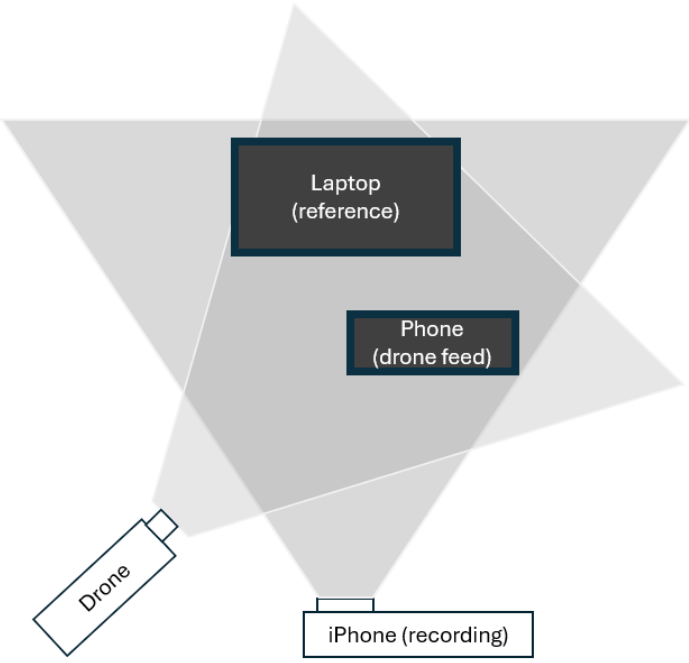


Figure 3.9: A drawing of the setup for a glass-to-glass delay measurement.

By recording all three screens and instantaneously changing the laptop display, the glass-to-glass delay can be calculated by counting the number of frames and multiplying by the frame time, i.e. the inverse of the frame rate. The glass-to-glass delay is written as an equation in equation 3.9 where Δt_{delay} is the total delay, n_{frames} is the number of frames, and t_{frame} is the frame time. The measurement error is simply the length of 1 frame, t_{frame}, or 0.004 s.

$$\Delta t_{\text{delay}} = n_{\text{frames}} \cdot t_{\text{frame}} \tag{3.9}$$

3.3.3 Drone control

To reach a position from which video can be captured the drone has to be sent to calculated and specified coordinates and perform necessary actions when reaching the position to be able to view the entire test area. To be able to perform these tasks. DJI's API related to programming your own waypoint missions(see Section 2.2.4) was used.

To easily handle all necessary actions in the best way and get a good code structure, a class called flightmanager was created to perform flight-related communication with the drone through the API. The skeleton for this was built with the help of the API guides on DJI:s website [28].

The approach when building the applications was to start at the beginning of the operation sequence represented in Figure 3.10 by building the main parts first such as applying coordinates from the user, uploading, starting, and stopping mimicking a simple and traditional DJI waypoint mission. After those features were in place and working, functions to make the waypoint mission work for the system were implemented such as waypoint actions and continuous footage of the coordinate and signaling a finished test to the drone combined with calling it home.

The activation and operation sequence is the bottom line of how the drone should be used in the system. The gray boxes represent things the drone or application executes and the text between the boxes commands to the drone.

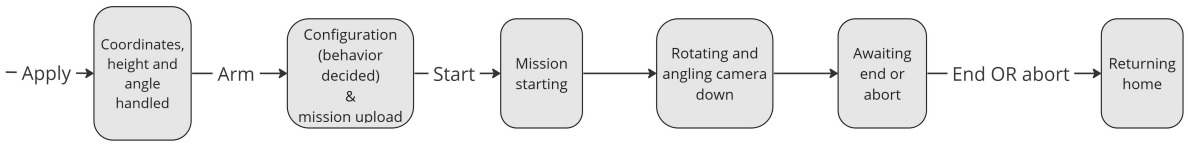


Figure 3.10: Drone activation and operation sequence

Testing of this activation and operation sequence was carried out on a soccer field close to Chalmers by entering a coordinate corresponding to the center of the field and the red cross in Figure 3.11. The starting point of the drone is represented with a green cross in the top right corner of the same image. By entering the center of the center circle it was possible to verify that the drone was actually flying to the correct position.

Figure 3.11: Soccer field for testing of drone actions. Red cross for waypoint coordinate. Imagery ©2024 Airbus, Lantmäteriet/Metria, Maxar Technologies, Map data ©2024

In the final setup similar to Figure 3.2, there is also a need for a more united control of the entire system preferably

from the ATOS communication software. This meant setting up a connection between the application and the computer running ATOS. This was made possible by WebSockets, see Section 2.2.5. A communication was set up between the application and the computer via a local connection on WiFi.

The connection sends strings between the devices which contain orders on what action to perform. A message handler was built in the application which can upload coordinates ready for execution or call different functions in the flightmanger class depending on what the message contains. This led to direct control between a remote computer and the drone.

3.4 Video streaming from phone to computer

Utilizing the DJI SDK , see Section 3.3.1, the live drone footage was displayed on the phone. The next step involved streaming this feed from the phone to a computer for object detection processing. While object detection is feasible on a phone, using a computer provides several advantages. First and foremost, implementing the object detection algorithm using YOLO, see Section 2.1.1, in Python is easier compared to integrating it into an Android application. Additionally, the computers we had available for the project had significantly higher computational power than the phones we had, enabling faster and more accurate detection of objects in the test area. Furthermore, a computer's larger display enhances monitoring clarity, which is crucial in our testing scenarios.

To get the video feed from the Android application to the computer, a third-party application called YoCast was used, which records the phone screen and streams it over a network. Although a more direct method of relaying the video feed through the phone is theoretically possible, this solution was chosen due to time constraints.

3.5 Object Detection

The object detection portion of the project was developed by the students at PSU. The model is based on YOLOv8 [39] and trained on a custom dataset. As such, only the sections of their work that are relevant to our work are discussed here.

3.5.1 Data collection and model training

The training data for the model is made up of 4591 images taken from a drone. Videos were recorded in both Gothenburg and State College of local streets using the drones of the respective groups. The PSU students had access to a DJI Mini 2 [22]. These were then manually labeled. In total, 2913 images have been labeled by the PSU students. As the model takes images as inputs and not videos, individual frames were extracted with an interval of 2000 milliseconds from the videos. This was done to provide a balance between the amount of data and training the model on too similar data. The model was trained by the PSU students. It was refined several times as more labeled images became available.

3.5.2 Model evaluation

The model is evaluated using the performance metrics from Section 2.1.2. The optimum threshold was calculated in the PSU report [40] and was used during the testing of the entire system. The model was also empirically tested on both live footage and pre-recorded footage to gauge how consistent the model was through longer sections of video, as the model only works on a frame-by-frame basis.

3.5.3 Coordinate mapping from drone footage

To enable the comparisons of positions from ATOS to what is detected by the drone, an algorithm is used which was developed by PSU. The center pixel of all bounding boxes in stored. Using the height of the drone, the field of view of the camera, and the GPS position of the drone, an estimated position of all the classified objects on screen is produced. These coordinates are then compared to the expected coordinates produced by ATOS and the discrepancy in meters is displayed to the user.

Chapter 4

Results

In this chapter, the results gathered from the project are presented. The developed product and responding results are divided into two steps, initially, an overview of the system is displayed with the final status of each subsystem. This section is further down accompanied by a more in-depth and detailed explanation and presentation of each subsystem's result. Below follows the aforementioned overview of the system. The gray squares represent a physical part of the project. The colors represent the status described in the image of the different modules.

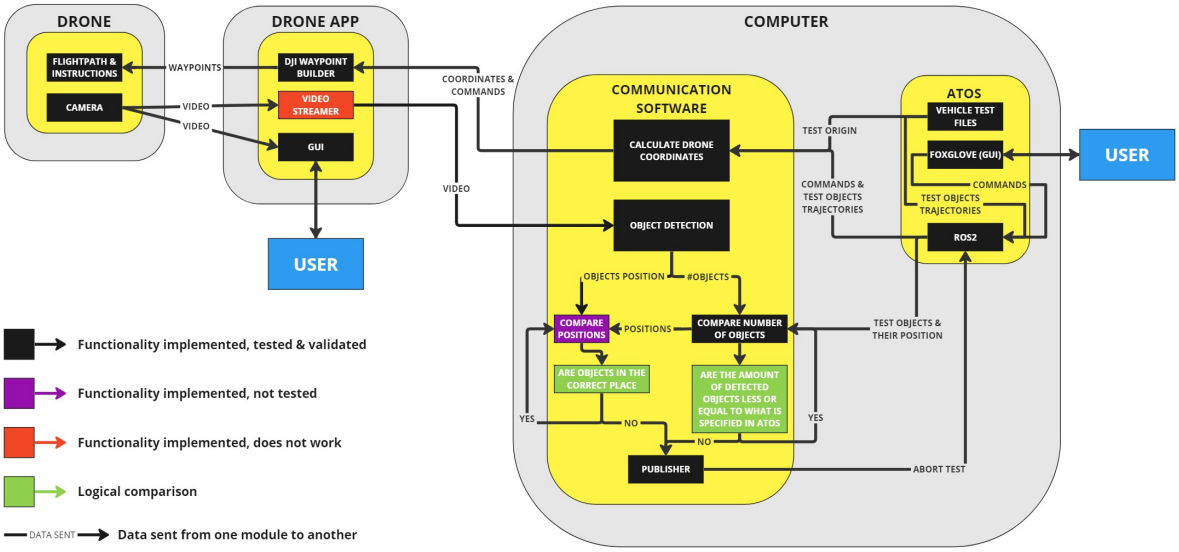


Figure 4.1: An overview of the system and its completeness.

As can be seen in Figure 4.1, a majority of the modules in each subsystem are fully operational which also results in a largely functioning subsystem and complete system. The non-functioning module is the video streaming module which impedes the computer from retrieving the camera footage from the drone by which it has no video frames to run object detection on. As this is a substantial problem for the end product, a workaround for this problem has been done which utilizes a third-party software for the streaming part.

The uncertainty in the module 'compare positions' is not the comparison itself, it is the position calculation in the object detection which has not been validated due to it requiring positional data from ATOS and the drone simultaneously which is only available when running a test at AstaZero. Since the module's late implementation in the project and the limited amount of visits to the test track mentioned in 1.4, it has not yet been tested and validated. A more detailed chapter on each subsystem and its corresponding results can be found below.

4.1 ATOS and communication software

By utilizing WebSockets, see 2.2.5, and the developed middleware, communication between the drone and ATOS was achieved. The custom communication software communicates via ROS2 and is able to retrieve the state of the test, see 3.2. The communication software has achieved two primary goals, calculating the position of the drone, and comparing the positions of the test objects in ATOS to those captured by the computer vision algorithm from the drone. To facilitate communication, the laptop acts as a WebSocket server.

4.1.1 Object coordinates from ATOS

The coordinates from the test objects were subscribed to and saved in a global variable. In Figure 4.2 the current longitude and latitude coordinate of an object with the ID 1 is printed.

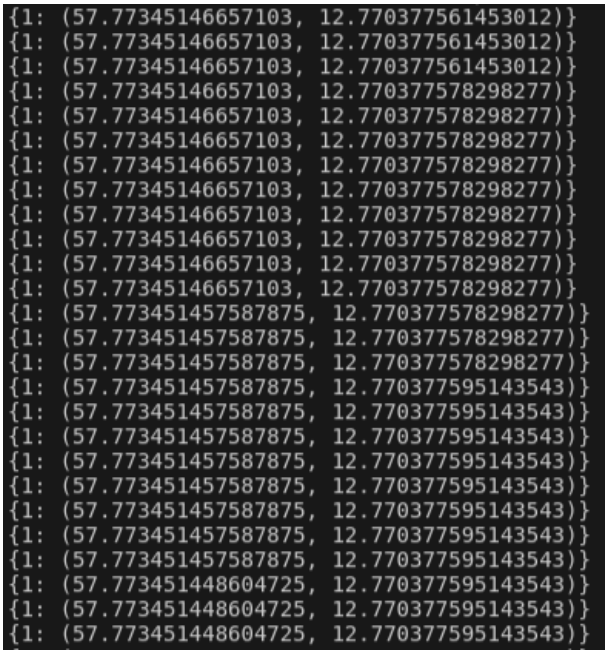


Figure 4.2: The live coordinates being printed

4.1.2 ATOS test states

ATOS state change instructions could be sent by the communication software, this was controlled by having the Foxglove GUI open while the states were being changed by the Python script, and seeing the state reported in Foxglove change. In a similar way, the way of reading the current ATOS state was verified to be working.

4.2 Drone positioning in 3D

The result of the two different height calculations mentioned in Section 3.2.3 is displayed in table 4.1 where the validation method mentioned in the same previously mentioned section was used in order to retrieve a given area from a certain height. This area was used as an input into the height calculations and the result is therefore the height outputted by both calculations. With images taken by the drone at a known height, the viewable area in the image is entered into Google Earth which returns the area. The area is entered into the two different calculations mentioned in the method.

Table 4.1: Comparisons of outputs from two different height calculations. The result of the calculation method currently in use in the project is under Method 1. The result of the other method based on [32] is shown under Method 2.

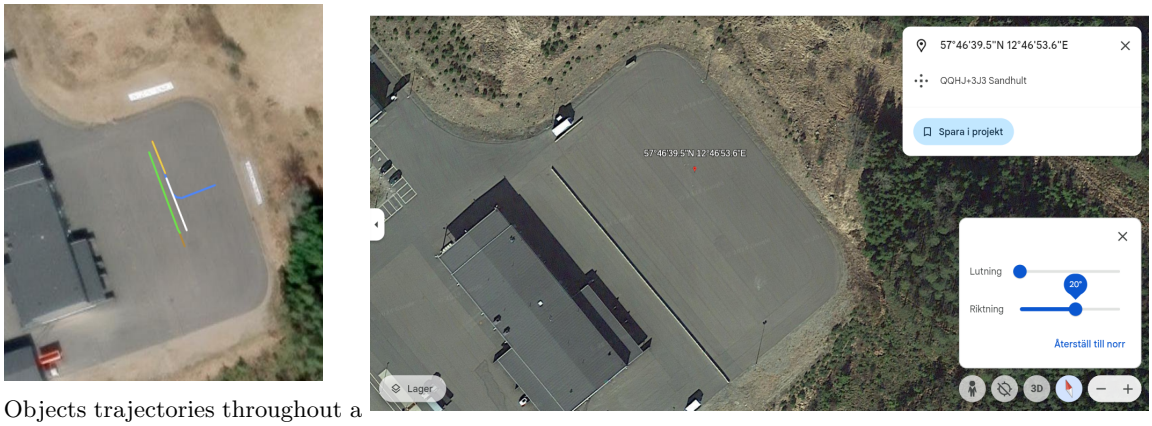
Area (m^2)	Known height (m)	Method 1 (m)	Method 2 (m)
445	20	20	18
730	25	26	23
1066	30	31	28
1455	35	36	33
1939	40	42	38
2987	50	52	47

As can be seen in Table 4.1 both of the methods return values similar to the truth. Method 1 returns values with less than or equal error as method 2 and was thereby chosen as the method to calculate the height.

The result of the drone positioning calculator was tested by comparing the calculated coordinate, angle, and height for the drone to the object trajectories in a given ATOS test, the trajectories are shown in Foxglove. The ATOS test in Foxglove is shown in Figure 4.4a and the calculated coordinates, height, and angle are shown in Figure 4.3. The resulting coordinate and angle was entered into Google Earth, which can be seen in Figure 4.4b.

```
[WARN] [1715332165.830490648] [atos publisher]: ATOS is not running, waiting ...
[WARN] [1715332166.840931649] [atos publisher]: ATOS is not running, waiting ...
[INFO] [1715332167.353521825] [atos publisher]: ATOS is running, waiting 10s to make sure that everything has started
[INFO] [1715332177.363952756] [atos publisher]: Publishing init signal
[INFO] [1715332178.371323725] [atos publisher]: Publishing connect signal
Lat: 57.777641323196896, Lng: 12.781542921067947, Alt: 23
Drone angle: 21
```

Figure 4.3: Result of the position calculation



(a) Objects trajectories throughout a test

(b) The calculated position and angle shown in Google Earth

Figure 4.4: The positional validation method used
Imagery ©2024 CNES/Airbus, Lantmäteriet/Metria, Maxar Technologies, Map data ©2024

As mentioned when the validation method was introduced in Section 3.2.3, this is not an optimal validation method because of the measurement errors, however the result of the translation and rotation of the drone looks reasonable.

4.3 Drone Application

The application's main functionality is hosting a hub for control of the drone and for transmitting information gathered when surveying the ATOS test rather than hosting any manual control over the system. A few interfaces have however been developed allowing control of the drone exclusively from the application and still stream live video to be analyzed by the computer.

The application is built using the DJI mobile SDK and the methods provided by their API and brings with it a certain number of functionalities that are used to control the drone and communicate with an external computer running ATOS.

4.3.1 GUI

In total, the application consists of 5 ”menus” with different purposes accessible according to the mapping in Figure 4.5 and are defined as follows:

1. Starting screen.
2. Main interface for active flying.
3. Coordinate settings to manually load coordinates and aircraft yaw.
4. Server settings to configure the WebSockets connection to the main computer.
5. Camerasettings to adjust the manual controls for the camera.

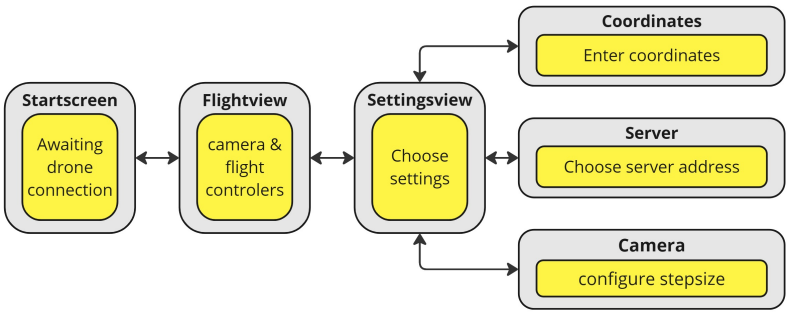


Figure 4.5: Mapping of menus

The main interface or flightview and the one used in the highest degree in the application is designed to ensure an easy and non-confusing experience for the user and is represented in Figure 4.6. The screen contains a camera view in the middle of the screen and buttonviews on each side. To avoid navigating through different menus while a test is carried out, all for active flight necessary buttons and status updates are presented on the side of the same screen. For increased clarity, all buttons and functionalities related to the camera is located on the left of the screen. The functions and buttons related to drone control are located on the right.

Figure 4.6: Main screen for the application

The status text in the upper left corner is updated every 0.750 seconds. It is used to signal to the user if the drone is ready to fly towards the desired coordinate. The colors used are green, yellow, blue, and white. When the drone is currently executing or moving by itself, the color of the text is green. When it is ready to start executing it is yellow, when ready to arm; blue, and in manual flight or some other state the text turns white. In the top right corner of the screen, there is a toggle switch that signals to the rest of the system if the user wants to allow remote control via the ATOS command line as an extra safety measure to ensure that the system parts will work as the user intends.

If one would like to manually configure the waypoint mission(see 2.2.4) the user has to input the coordinates into coordinates settings which interface is shown in Figure 4.7.

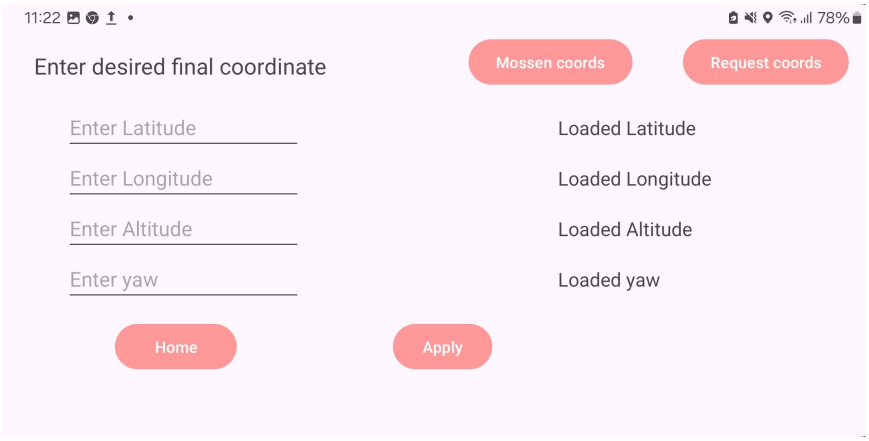


Figure 4.7: Interface for coordinate settings.

The interface consists of four input fields where the user can configure the GPS-coordinates, altitude, and heading in degrees from north. In this view, there is also the possibility to fetch coordinates from the computer running the communication software (Figure 4.1) via a WebSockets server if one is connected and has initialized an ATOS test. If the button is pressed, the coordinates, required height and jaw show up in the fields and the user can press apply.

To configure the connection to a WebSockets server, the interface for server settings is used and shown in Figure 4.8. This is a simple view with input fields for choosing the IP-address and port for the connection. When the user presses connect, a confirmation is shown below the input fields. Once that happens, the user can continue to use that application as before, but can now also fetch coordinates and allow remote control of the drone from the computer.

Figure 4.8: Interface for server settings.

The last menu in the application is used to configure the manual camera controls in the flight view and is represented in Figure 4.9. Specifically, the things adjustable here are the stepsize in degrees of the camera when pressing either rotate right or rotate left.

The purpose of this is to have a better opportunity to get a good alignment of the test area and to be able to capture as much of the test as possible from the lowest altitude possible.

4.3.2 Video streaming from drone to phone

The video feed from the drone's camera is retrieved via the DJI SDK. The glass-to-glass delay outlined in Section 3.3.2 was measured. It takes 57 frames for the screen update to be reflected on the phone, with a frame time of $\frac{1}{240} \approx 0.0042$ s. Using equation 3.9, this results in a delay of 0.237 ± 0.004 seconds.

Figure 4.9: Interface for camera settings.

4.3.3 ATOS communication and WebSockets

To position the drone in the test scene and to enable control of the drone from a computer, the phone and computer communicate via WebSockets, see Section 2.2.5. With the optimal coordinate calculated by the ATOS communication software presented in Figure 4.1, the coordinates are transmitted to the phone. First, the phone connects to the computer acting as a server. Secondly, the user requests the coordinates from the phone in the user interface. Thirdly, the coordinates are sent back to the phone and from there the phone stores them and makes them available for the flight plan in the next section.

WebSockets communication between the phone and computer is also able to control the start and end of the drone. To enable remote control from the computer, the previous step of coordinate loading must be completed and the user must enable remote control in the application using the toggle in Figure 4.6. When the user sends the take-off, arm, or end command from the computer, a keyword is transmitted via WebSockets to the phone. When the phone receives such a keyword and the aforementioned remote flight toggle is activated, the drone will perform this action. This is performed with calls to the same methods that are called when the buttons in the UI are pressed.

4.3.4 Drone control

In order to control the drone, the DJI API is utilized with its method to build and execute waypoint missions together with methods related to the general state of the drone. The system takes input from either the user manually within the application or remotely from a computer running ATOS via WebSockets described in Section 4.3.3. The system related to configuring missions and flying looks like represented in Figure 4.10.

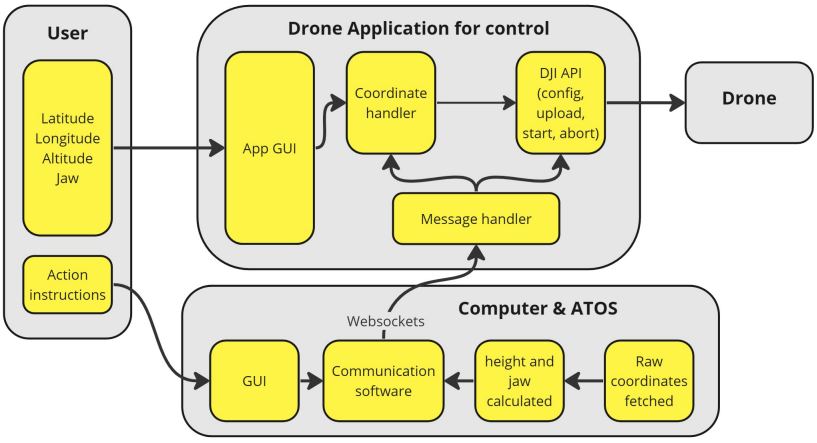


Figure 4.10: System overview dronecontrol

There are two ways to use the application. One way is to manually insert coordinates in coordinatessettings as shown in Figure 4.7 to the application in the GUI, arm, start, and end when done in the main flightview.

The second way is to control the drone through the computer and its communication software. This requires a setup WebSockets communication and that the computer currently runs ATOS in order to have positions to fetch and subscribe to ROS2-topics, see Section 2.2.1. Here the application receives instructions and messages through a message handler within the application that calls different functions and methods depending on the message it receives. The functions called are the same as those being called when using the application manually meaning that it is possible to combine control methods between remote and manual.

Regardless of which way the user wants to control the drone, the same steps have to pass in order for the drone to take off and return home when done. Those steps are described below:

1. Upon receiving coordinates either from a text-input field or via WebSockets connection, the coordinate handler checks that the coordinates are valid so that the application does not throw any errors. A valid coordinate means that they are actual numbers, the altitude surpasses 5 m above ground, and that the yaw is within $[-180, 180]^\circ$ required by the API. After the values of latitude, longitude, altitude, and yaw have been approved, they are loaded into the flightmanager class and await usage.
2. When receiving an arm-message either from the flightview via a button or a remote command the flightmanager performs a few checks to ensure the drone is ready to perform a mission. These checks include checking a sufficient battery percent above 20% is fulfilled and an existing GPS signal. When these checks have passed the flightmanager creates a "dummy waypoint" 10 m above the starting position of the drone. The reason behind this is that DJI:s waypointmissionbuilder does not allow less than two waypoints in a mission which produces an error. The configuration continues in the next step.
3. The main waypoint is created on the position of the coordinates loaded in the flightmanager in step 1. After this, the application configures the mission with the desired behavior of the drone pre-, during- and post-waypoint. These behaviors are as follows:
   - Flightspeed is set to a constant of 6 m/s throughout the mission.
   - Changing the yaw of the aircraft at the primary waypoint to optimize the window angle to capture the entirety of the test scenario. The yaw value is what has been input, if nothing, then the drone will head 0° corresponding to direct north at the waypoint.
   - Upon reaching the primary waypoint, angle the camera straight down to capture what happens on the ground.
   - Stay put at the final waypoint until an end command is sent to the drone either directly from the GUI on the phone or from the ATOS interface signaling that the test is over.
4. After the mission has been configured in stages 2 and 3 it is uploaded to the drone and awaits a start command from the user.
5. When receiving a start command either from the user pressing a button in the application or from the ATOS commandline via websockets the drone takes off and starts flying towards the test center.
6. When hovering over the test area, a continuous video stream is kept to the phone, and the drone stays put until an end or abort command is sent. When this command is received, the drone returns home to the starting position, angles the gimbal up, and lands awaiting the upload of a new mission.

This series of actions can be interrupted at any time from the application by the abort button by which the drone stops and starts hovering in the current position and manual flight is initiated and the user can bring the drone home manually with the controller. Alternatively, the user can press end at any time which aborts any running mission and sends the drone back to the starting position.

4.4 Video streaming between phone and computer

As mentioned in Section 3.4, there was no successful development of a custom solution to stream the feed of the drone's camera to a computer. The YoCast application was used to stream the entire screen over a local network connection to the laptop. The video feed was then used as an input to the model. YoCast is an off-the-shelf Android application that can record the stream and enables users to receive the screen capture over a local network connection.

The glass-to-glass delay described in Section 3.3.2 is varied when using YoCast. On average, the delay varied from 1 to 4 seconds when viewing the stream in a web browser. This latency is low enough that it can be used in the project to perform real-time object detection. However, the usage of the application has been unstable. It is sometimes necessary to restart YoCast several times to get it to start and display on the computer.

4.4.1 Attempted solutions

Below are the methods that were explored and led to the usage of YoCast in order.

- **DJI's SDK:** The SDK from DJI offers tools to access and stream the drone's camera feed from an Android device to a computer. Although the SDK supports streaming protocols such as the Real-Time Messaging Protocol (RTMP) [41], a notable issue is the approximately 30-second delay introduced by RTMP, which is not suitable for applications where real-time observation is crucial.
- **WebSockets:** Using WebSockets, 2.2.5 a connection between two computers for sharing a camera feed was successfully established. However, attempts to establish a connection between the Android device and the computer using WebSockets with the DJI API were unsuccessful.
- **Sockets:** Sockets 2.2.6 enable direct communication between devices over a local network. After substantial effort, a textual data connection was successfully established between the phone and a computer using a basic chat application developed in Android Studio 2.2.3 for a connection establishment test. However, transmitting video data posed a greater challenge. Sockets are a basic, low-level mechanism for data communication, meaning that everything apart from raw data input and output, such as encoding and decoding data into a suitable format and managing error detection and correction, must be handled manually. Unlike text data, video data transmission requires greater network capacity and real-time processing, and involves a more complex application, all of which demanded additional effort and time.
- **WebRTC:** Another option explored involved code previously developed by AstaZero that used WebRTC 2.2.7 to establish a connection via a live server. WebRTC enables direct, peer-to-peer communication for streaming audio, video, and data without the need for an intermediary. Initially, this approach allowed for the sharing of camera feeds between devices, successfully facilitating the transmission of the camera feed from a phone to a computer. Further exploration enabled the sharing of screens between two connected computers. However, limitations were encountered when attempting to share the screen from a phone to a computer, due to the lack of support and other restrictions in Android browsers that prevent direct sharing of the Android screen with a computer [42].

4.5 Object detection

Receiving the video stream from the phone, a program using an AI model based on the YOLOv8 architecture is used for making inferences and detecting objects in the video frames. The program also provides a live annotated view of the video stream, allowing the user to visually follow the detected objects.

The model has been trained on two classes of objects, persons and vehicles. Tests of the model for both classes give it an F1 score 2.1.2 of 97% for a confidence threshold of 0.484, which exceeds the target of 90% that was set in our project plan.

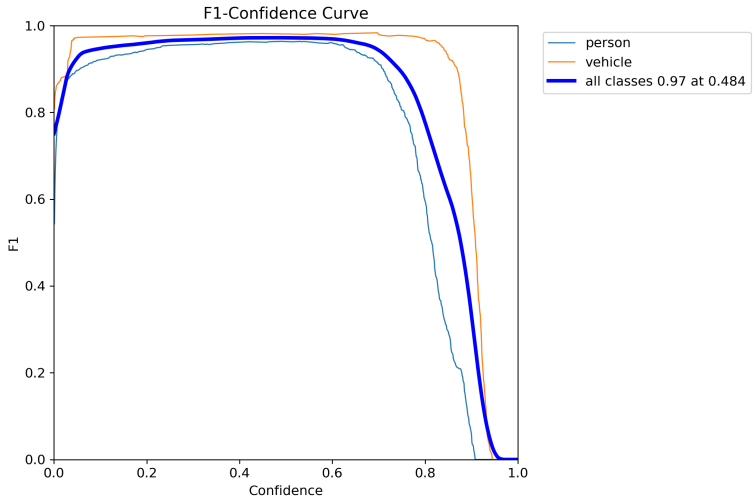


Figure 4.11: F1-Confidence Curve. Source: [40]. Reproduced with permission

The model also accomplished a quite high mAP. In Figure 4.12 it is shown that the mAP50 ended up on a score of 0.9842 after 100 epochs of training. The mAP50-95 ended up on a score of 0.74889.

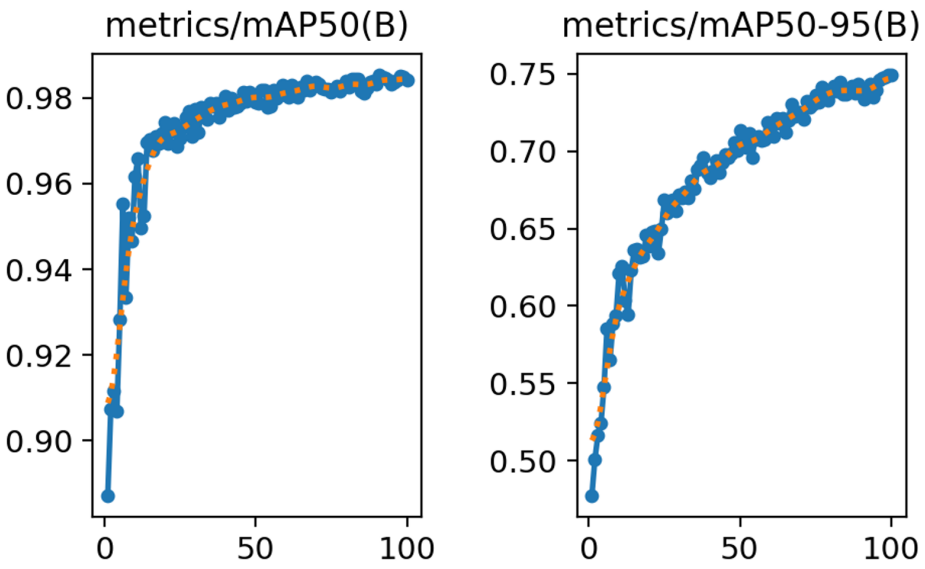


Figure 4.12: mAP values for the model over 100 epochs of training. Source: [40]. Reproduced with permission

By running the model on video clips and manually reviewing the data, it is possible for the model to loose track of a detected object in the scene, only for it to be regained by the model a few frames later. The model only runs on individual frames and does not keep track of objects over multiple frames. Empirical testing does however show that there are very few situations where the model overcounts the number of objects in the frame.

When using the live video from the drone, there are significant performance delays that grow the longer the video stream is viewed. The delay varies from run to run. In some cases, it is as low as one second while it sometimes grows as large as 5 seconds. It is also observed that the delay grows over time when the model is running.

The data received from calculating the coordinates of the objects in the video was compared in real time to the coordinates that was retrived from the ATOS object coordinate subscriber mentioned in Sections 4.1.1 and 3.5.3. This returns a distance between the detected object in the video to the closest object expected from ATOS.

4.6 System operation process

After discussing the different blocks that build the system, here's a guide on how to operate the final system:

1. **Connect the Phone to the Drone:** Connect the phone to the controller using a USB cable. Ensure the connection is established by confirming the live video feed from the drone is visible on the phone screen.
2. **Stream the Screen Feed to a Computer:** Install and open the YoCast application on the phone. In the app, start streaming the phone screen feed (showing the drone's video) to a computer, meanwhile, make sure to run the Python script that utilizes a cv2 module to receive the video data and view the live stream from the drone.
3. **Start the Communication Software:** Launch the communication software on the computer to establish a connection with the phone.
4. **Retrieve the Coordinates of Test Objects:** Open the Android application on the phone. Click the "Request coords" button to obtain the coordinates of the test objects from ATOS and verify that the coordinates are successfully received and displayed in the application.
5. **Navigate the Drone to the Test Field:** On the computer, send an arm and a start message to the drone. Input the test field coordinates and calculated height position into the software and command the drone to navigate to the test field and ensure it reaches the correct latitude position.
6. **Start the Test from ATOS:** In the ATOS system, initiate the test to start the automated vehicles. Ensure the vehicles begin driving according to the predefined test parameters.

7. **Supervise the Test with YOLO Object Detection:** On the computer, view the drone's camera feed through YoCast. Start the YOLO object detection software to analyze the video feed, supervise the test and monitor the detection results in real-time to ensure accurate object identification and tracking.

Chapter 5

Discussion

This chapter discusses and evaluates the different parts of the system as a whole. This concerns the degree of success and where the current state of the product lies. The chapter also discusses aspects of the product consisting of possible future work within the project and social and ethical aspects related to the product.

5.1 ATOS and communication software

The communication with ATOS works. It was a challenge to find the necessary information and getting it to work, but it will simplify further work with the project or work on similar projects.

The position of all of the objects in a test can be updated continuously, the test states can be read and changed. It is also very possible to compare the number of cars and people detected by the camera to the expected number of cars and people given by ATOS. It was shown that the number of vehicles captured by the object detection can be compared to the number of objects expected and a warning can be printed when it does not match. A solution for comparing the distance between the position captured by the camera and the position given by the test objects was developed but not thoroughly tested.

Verifying that sending the *abort* signal to ATOS worked was integral for the project. This was verified to be working. This means that the autonomous tests could theoretically be interrupted by the software if something was out of place.

5.1.1 Drone positioning in 3D

Due to several sources of error in the measurements, primarily the human error of fitting the edges of the drone's picture onto Google Earth's map, the results obtained from the verification of the drone placement algorithm are not precise, see Section 3.2.3. However, an exact measurement was not required for this validation and therefore this method was deemed sufficient. As previously mentioned Google Earth was also used in order to validate the position and orientation calculations. The best validation method for this would be to see how the drone positions itself during a live test, but the limited opportunities to visit the test track mentioned in Section 1.4 meant this was not a viable option. With the tools available at the time, the second best validation method was to review the footage and position of the drone and the test objects gathered from the previous visit and use this to validate.

5.2 Android application

In evaluating this project, it is clear that the Android application that was developed to control the drone was the most successful part of the project.

This echoes the results achieved by an AstaZero-sponsored bachelor's thesis project from last year [43]. Unfortunately, we were unable to re-use last year's application due to dependency issues and lack of documentation. This meant that the start of the project consisted of essentially recreating the application that was created last year from scratch.

A rewrite of the application was not without merit, as it allowed for the creation of a minimal application that could be properly documented from the beginning. This leads to two main advantages. Firstly, it means it was easier to extend the application during the development process and introduce new features. Secondly, it allows anyone else looking to extend the project to more easily develop the application. This could be both at AstaZero, as well as for future bachelor theses. The drone used in this project, the DJI Mavic 2 Enterprise, supports version 4 of the SDK [44]. The latest version of the SDK is version 5, and this means that the latest features are not available for use with the drone available in this project.

WebSockets are used to communicate between the computer and phone in this project. WebSockets were primarily chosen as they were an alternative for streaming the video feed as well. The idea was to utilize just one connection and send the video data and commands in parallel. The WebSockets protocol supports the transmission of arbitrary binary data [23], so it could have worked in theory. This would likely have led to other problems, most notably some kind of processing on the computer end to rebuild the data stream. When the decision to abandon WebSocket-based streaming was made, the code base for WebSocket-based messaging had already been implemented, which meant its use was continued. The functionality of the WebSockets API and its focus on low-latency and overhead reduction is not necessary for this project but there are good libraries available for both Android and Python which means WebSockets are easy to work with.

5.3 Video streaming

When evaluating the project, it was found that the video streaming section was the part of the project where the least success was seen in fulfillment of the set goals. Despite several attempts, no custom solution was developed. In part, this was because a functional yet unoptimal solution was available, and a decision was made to use the limited time to develop other features.

5.3.1 Usage of YoCast in the project

The chosen solution of a commercial off-the-shelf application YoCast described in Section 4.4 presented several challenges and limitations in the project. Firstly, it introduced the delay of a few seconds mentioned in Section 4.4. While this delay was acceptable, it was not optimal as it meant that any abort signal would also end up delayed by that much time. In a test scenario where a hypothetical test vehicle is moving at 60 km/h, the vehicle would end up 17 meters "ahead" of the safety system monitoring the test. This would either mean that the system would not react on the sudden onset of a dangerous situation, or that larger margins of safety would be required. This would in turn lead to more false positive reactions from the system.

Secondly, lag was induced which would also result in the footage being temporarily delayed further before catching up. This created further delays for the object detection software.

Thirdly, as the video stream is a screen capture, rather than a stream of the camera feed, there is an unnecessary loss of quality and more data than necessary is transmitted. In an ideal solution, only the raw video feed of the drone's camera would have been transmitted. This would have been attained through the DJI API and then transmitted in some way. Not only would this cut out streaming unnecessary screen info that must be cropped before using as input to the model, but it would have allowed streaming of the video feed in a higher resolution with obvious advantages for object detection performance.

5.4 Object detection

The model that was developed is performing to such a degree that it can be used as a basis to perform the objective, namely to detect vehicles during a test and compare them to the data from ATOS. As mentioned in section 4.5, the model will sometimes fail to classify an object on a frame-by-frame basis.

The consequence of this is that either the model will feature significant false-positives and cause the system as a whole to see a mis-match in the number of vehicles just because of a computer vision problem. This can be compensated for by only activating if the model detects a problem over a period of time. However, this introduces delays from when a dangerous situation occurs to when an alert is issued.

5.5 Future work

In the future, the goal is to implement this system at AstaZero so that they can use the system in actual testing. However, this would require further development and work in multiple areas of the system as the preferred final

version is not yet achieved. To be a product appropriate for active use at AstaZero's test track a few areas need more work. These areas include working video streaming from the phone application to a computer, a more versatile object detection software, a more sophisticated design behind the decision of what is a hazardous situation, and visuals and controls incorporated into Foxglove Studio described briefly in Section 2.2.2. These points are explained here.

The main focus for development in the future for this system would be to produce a good and working method for video streaming between the phone running the drone application and a computer running the communication software. At the end of the project, that is solved via a third-party application described in 4.4. An in-house developed solution would be beneficial to allow for adaptability, and the possibility to develop and adapt every part of the system as well as a much more compact and easy-to-use system. Having a solution that is also fast would also be preferred to increase safety by swiftly detecting dangerous situations.

In the future, the model for object detection could be diversified and improved to work at more heights and for a broader selection of classes. In the current system, the weights in the object detection software are close to only being trained on images from a height of about 40m. This leads the model to perform worse at higher altitudes. This is however less of a problem in terms of development and more of a work hours issue which implies filming more footage and running more training. It is worth noting that at a substantial height, detecting people at all would be difficult. To solve this problem an alternative solution rather than exclusively surveying the test area statically from straight up would be beneficial. Methods for this could be more extensively worked on and perhaps look into a dynamic drone moving in circles around the test area at a lower height represented in Figure 5.1.

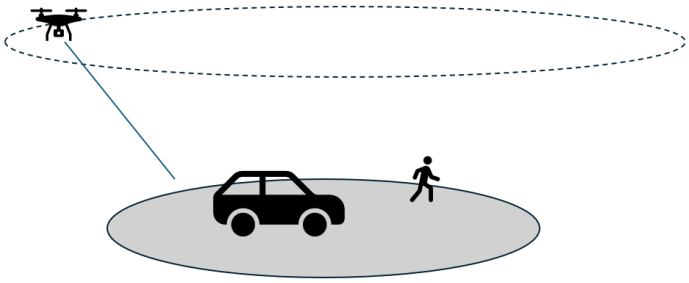


Figure 5.1: Example of an alternative solution for large test scenarios

In terms of a broader selection of classes, the model is currently only detecting people and vehicles. To keep working on training the model on for example wildlife or bicycles would be beneficial to have the whole system become more comprehensive and make better decisions with regards to what is and what is not appearing in the test scenario.

The decision-making can in its current form count different types of objects and detect discrepancies in terms of the amount of objects as well as to some degree the distance to one another. To also have the system be able to predict expected trajectories of misplaced or unauthorized objects could create nuance as to how severe of an action to take in response to the discrepancy. This is an issue that has been discussed throughout the project, however with time as a limitation it was never realized. That said it would be a certainly interesting development for this system.

A constant request from AstaZero has throughout been to incorporate as much of the system controls and messages into Foxgloves Studio from which the ATOS test scenario is initialized and controlled. Things that could be incorporated here in the future could be visualizations of when a discrepancy on the test site occurs resulting in a clearer user experience. Additionally, incorporating controls such as connecting, loading coordinates, arming, and starting the drone on regular ATOS commands like arm ATOS or as separate drone-related buttons would greatly benefit the user experience and minimize the extra steps required to implement this system. This might also be a great addition to ATOS as a product.

5.6 Social and ethical aspects

There are multiple relevant ethical aspects to consider relevant to this project that one has to keep in mind. These issues include both the matter of privacy and the risk of injury.

One ethical issue to consider is as mentioned before related to privacy. Both during development and the final product will utilize automated camera surveillance from an automated drone. This can be considered an invasion of privacy if something goes wrong and non-intended targets get captured by the camera.

During actual operation, the drone flies at such a height that no details of people will be visible as this system doesn't detect any identification of individuals. The system only records the type of object and approximate location while using object detection. However will structures of for example buildings still be possible to recognize.

Between 2015 and 2020, there were approximately 3700 hospital visits in the U.S. because of drone-related injuries [45]. This highlights the potential danger of using a drone in close proximity to people. When operating the system, care must therefore be taken to operate the drone safely. However, by complying with regulations regarding drone usage [46], these risks can be minimized.

In the finished state, the impact of the ethical considerations is dampened. This is because it will only be used in a closed environment. This means that any person is aware of the recording and analysis taking place decreasing the risk of invading privacy. As the test area already must be cleared before the test can start, the likelihood of any injuries is small as well as everyone intended to be nearby will already be aware of the situation around them.

It is also worth noting the potential for misuse and abuse of the system, primarily related to camera surveillance. An unknown drone monitoring you, both knowingly and unknowingly is a big invasion of privacy. Even with this potential ethical issue, the pros are so much stronger than any issues that the project will proceed.

Chapter 6

Conclusion

This report outlines the development of a software solution using a drone and computer vision to monitor an autonomous vehicle test and alert the test supervisors in the event of a hazardous situation. A set of programs have been developed to automatically calculate the optimal drone position based on the expected trajectories of the test vehicles. The drone can then autonomously position itself at this location and monitor the test. A computer vision model based on YOLOv8 was produced that had an F1 score of 97% with a confidence threshold of 0.484.

The results of this project indicate that it is possible and advantageous to deploy the developed solution to monitored tests, but that further development is necessary for a more complete monitoring of vehicle tests.

To improve the performance of the proposed solution, it would be necessary to reduce the video latency. It is also recommended that the object detection model is run either on hardware with more computing power, or that the model is improved. Further research in this topic could explore a more formal definition of what constitutes a dangerous situation and how they could be classified. Another proposed development is the implementation of a custom video streaming solution between the phone and computer.

References

[1] *Driver assistance technologies*, National Highway Transportation Safety Administration. [Online]. Available: https://www.nhtsa.gov/vehicle-safety/driver-assistance-technologies (visited on 05/09/2023).

[2] M. Galvani, "History and future of driver assistance," *IEEE Instrumentation Measurement Magazine*, vol. 22, no. 1, pp. 11–16, 2019. DOI: 10.1109/MIM.2019.8633345.

[3] *About us*, AstaZero. [Online]. Available: https://www.astazero.com/en/about-us/ (visited on 05/10/2024).

[4] *Test stie*, AstaZero, May 2023. [Online]. Available: https://www.astazero.com/en/tracks-facilities/test-site/ (visited on 05/09/2023).

[5] *Our services*, AstaZero, Apr. 2023. [Online]. Available: https://www.astazero.com/en/our-services/ (visited on 05/09/2023).

[6] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, vol. 111, no. 3, pp. 257–276, 2023. DOI: 10.1109/JPROC.2023.3238524.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010, Cited by: 12658; All Open Access, Green Open Access. DOI: 10.1007/s11263-009-0275-4. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-77951298115&doi=10.1007%2fs11263-009-0275-4&partnerID=40&md5=a367d0d47ac02a316e9ea5f1104a49e1.

[10] S. Ma, H. Lu, J. Liu, Y. Zhu, and P. Sang, "Layn: Lightweight multi-scale attention yolov8 network for small object detection," *IEEE Access*, vol. 12, pp. 29 294–29 307, 2024. DOI: 10.1109/ACCESS.2024.3368848.

[11] S. N. Moutsis, K. A. Tsintotas, I. Kansizoglou, S. An, Y. Aloimonos, and A. Gasteratos, "Fall detection paradigm for embedded devices based on yolov8," in *2023 IEEE International Conference on Imaging Systems and Techniques (IST)*, 2023, pp. 1–6. DOI: 10.1109/IST59124.2023.10355696.

[12] S. Mittal, "A survey on optimized implementation of deep learning models on the nvidia jetson platform," *Journal of Systems Architecture*, vol. 97, pp. 428–442, 2019, ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2019.01.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762118306404.

[13] K. Rungsuptaweekoon, V. Visoottiviseth, and R. Takano, "Evaluating the power efficiency of deep learning inference on embedded gpu systems," in *2017 2nd International Conference on Information Technology (INCIT)*, 2017, pp. 1–5. DOI: 10.1109/INCIT.2017.8257866.

[14] S. Hossain and D.-j. Lee, "Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with gpu-based embedded devices," *Sensors*, vol. 19, no. 15, 2019, ISSN: 1424-8220. DOI: 10.3390/s19153371. [Online]. Available: https://www.mdpi.com/1424-8220/19/15/3371.

[15] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2020, pp. 237–242. DOI: 10.1109/IWSSIP48289.2020.9145130.

[16] H. Zhang, A. Rogozan, and A. Bensrhair, "An enhanced n-point interpolation method to eliminate average precision distortion," *Pattern Recognition Letters*, vol. 158, pp. 111–116, 2022, ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2022.04.028. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016786552200126X.
[17] *Ros wiki*, Open Robotics, Jan. 2024. [Online]. Available: https://wiki.ros.org/ROS/Introduction (visited on 01/29/2024).
[18] B. Gerkey, *Why ros 2?* Open Source Robotics Foundation, Jun. 2014. [Online]. Available: https://design.ros2.org/articles/why_ros2.html (visited on 05/07/2024).
[19] ATOS, *Summary*, AstaZero, Aug. 5, 2023. [Online]. Available: https://atos.readthedocs.io/en/latest/.
[20] *Meet android studio*, Google, Apr. 30, 2024. [Online]. Available: https://developer.android.com/studio/intro (visited on 05/02/2024).
[21] *Mavic 2 enterprise series*, DJI. [Online]. Available: https://www.dji.com/se/mavic-2-enterprise (visited on 02/05/2024).
[22] *Dji mini 2 specs*, DJI. [Online]. Available: https://www.dji.com/se/mini-2-se/specs (visited on 05/07/2024).
[23] A. Melnikov and I. Fette, *The WebSocket Protocol*, RFC 6455, Dec. 2011. DOI: 10.17487/RFC6455. [Online]. Available: https://www.rfc-editor.org/info/rfc6455.
[24] *An introductory 4.4bsd interprocess communication tutorial*. [Online]. Available: https://docs-archive.freebsd.org/44doc/psd/20.ipctut/paper.pdf (visited on 05/06/2024).
[25] *Python 3.12.3 documentation, socket programming howto*. [Online]. Available: https://docs.python.org/3/howto/sockets.html (visited on 05/06/2024).
[26] *Real-time communication for the web*, Google. [Online]. Available: https://webrtc.org/ (visited on 05/07/2024).
[27] M. Shacklett and S. Johnson, *Webrtc (web real-time communications)*, TechTarget. [Online]. Available: https://www.techtarget.com/searchunifiedcommunications/definition/WebRTC-Web-Real-Time-Communications/ (visited on 05/07/2024).
[28] *Mobile sdk v4*, DJI. [Online]. Available: https://developer.dji.com/mobile-sdk-v4/ (visited on 05/07/2024).
[29] *Rclpy documentation*, Open Robotics, Jan. 2019. [Online]. Available: https://docs.ros2.org/latest/api/rclpy/index.html (visited on 05/07/2024).
[30] *Ros2 documentation*, Open Robotics, Jan. 2022. [Online]. Available: https://docs.ros.org/en/iron/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html (visited on 05/07/2024).
[31] *Atos github page*, RISE, Jan. 2024. [Online]. Available: https://github.com/RI-SE/ATOS (visited on 05/07/2024).
[32] R. S. Lunazzi, "Photogrammetric survey of heritage objects: An experimental protocol for optimizing camera placement for aerial surveys," in *2018 3rd Digital Heritage International Congress (DigitalHERITAGE) held jointly with 2018 24th International Conference on Virtual Systems Multimedia (VSMM 2018)*, 2018, pp. 1–7. DOI: 10.1109/DigitalHeritage.2018.8810046.
[33] *Ros2 bag documentation*, Open Robotics, Jun. 2024. [Online]. Available: https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Recording-And-Playing-Back-Data/Recording-And-Playing-Back-Data.html (visited on 05/10/2024).
[34] *Ros2 bag github*, Open Robotics, Jun. 2024. [Online]. Available: https://github.com/ros2/rosbag2 (visited on 05/10/2024).
[35] DJI. [Online]. Available: https://developer.dji.com/document/e26386b3-3d4e-490d-bec3-785f078045b8 (visited on 05/09/2024).
[36] *Camera application*, DJI, Jun. 28, 2020. [Online]. Available: https://developer.dji.com/document/06724d27-23cf-4741-b128-fc17d2891981/ (visited on 05/07/2024).
[37] C. Bachhuber and E. Steinbach, "A system for high precision glass-to-glass delay measurements in video communication," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 2132–2136. DOI: 10.1109/ICIP.2016.7532735.
[38] *Iphone 11 pro - technical specifications*, Apple. [Online]. Available: https://support.apple.com/en-us/111879 (visited on 05/07/2024).
[39] D. Reis, J. Kupec, J. Hong, and A. Daoudi, *Real-time flying object detection with yolov8*, 2023. arXiv: 2305.09972 [cs.CV].
[40] H. Chaudhry, E. Salvatori, T. Vittitow, and X. Wenhao, "Drone platform for safety in testing team highflyers final report," Unpublished, PSU student report.

[41] *Livestreammanager*, DJI. [Online]. Available: https://developer.dji.com/api-reference/android-api/Components/LiveStreamManager/DJILiveStreamManager.html (visited on 05/08/2024).
[42] *Screen capture api*, Mozilla, Aug. 19, 2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Screen_Capture_API (visited on 05/08/2024).
[43] J. Gustafsson, C. Oresten, T. Sallén, M. Sanderöd, L. Smedeman, and A. Toresson, "Automate the video documentation of a EuroNCAP test by use of drones," Chalmers University of Technology, 2023.
[44] *Mobile sdk v4*, DJI. [Online]. Available: https://developer.dji.com/mobile-sdk-v4/ (visited on 05/10/2024).
[45] S. Gorucu and Y. Ampatzidis, "Drone injuries and safety recommendations," *EDIS*, vol. 2021, 3 Aug. 2021. DOI: 10.32473/edis-ae560-2021.
[46] *Drönare*, Transportstyrelsen, Jan. 2024. [Online]. Available: https://www.transportstyrelsen.se/sv/luftfart/luftfartyg-och-luftvardighet/dronare/ (visited on 01/29/2024).

Appendix A

User requirement specification

Table A.1: The complete user requirements specifications as set in the beginning of the project.

Criteria	**Control method**	**Target value**	**Requirement / Desire**
Accurate object detection	Mean average precision	70%	R
Accurate object detection	Mean average precision	90%	D
Signal the test supervisor	Empirical testing	Yes	R
Stop automated vehicle test	Empirical testing	Yes	D
Different degrees of severity alarms	Empirical testing	Yes	D
Test area coverage	Measure area with camera	500 m^2	R
Test area coverage	Measure area with camera	>1000 m^2	D
Test duration	Empirical tests of max flight time	30 min	R
Accurate object detection while moving at 30km/h	Average accuracy of correct detection	70%	D
Live video coverage of test	Empirical tests	Yes	R
High usability with a well thought through UX	User survey	>70% satisfied	D

Appendix B

Code

B.1 Python code for subscribing to test object coordinates

```
def coordinate_callback(self, msg, id):
        with globals.mutex_atos_lock:
            globals.atos_object_coordinates[id] = (msg.latitude, msg.longitude)

def start_coordinate_subscriber(self):
    if len(globals.atos_object_coordinates) == 0:
        self.get_object_ids()
    coordinate_subscribers = []

    for id in globals.atos_object_coordinates.keys():
        topic = '/atos/object_' + str(id) + '/gnss_fix'

        coordinate_subscribers.append(
            self.create_subscription(NavSatFix, topic,
            lambda msg: self.coordinate_callback(msg, id), self.QOS))
```

B.2 Python code for getting test object trajectories

```
def get_object_traj(self, object_id):
        object_traj_req = GetObjectTrajectory.Request(id=object_id)
        future = self.get_traj.call_async(object_traj_req)
        rclpy.spin_until_future_complete(self, future)
        try:
            response = future.result()
        except Exception as e:
            self.get_logger().error('Service call failed %r' % (e,))
            return None

        if not response.success:
            self.get_logger().error('et origin service call failed for object %u', id)
            return None
        else:
            trajectories = []
            for point in response.trajectory.points:
                trajectories.append(Coordinate(point.pose.position.x,
                                    point.pose.position.y,
                                    point.pose.position.z))
            return trajectories
```

B.3 Python code for getting the control state of ATOS

```
class ExampleNode(Node):
    def __init__(self) -> None:
        super().__init__('example_node')
        self.OBCSTATES = {
                0: "UNDEFINED",
                1: "IDLE",
                2: "INITIALIZED",
                3: "CONNECTED",
                4: "ARMED",
                5: "DISARMING",
                6: "RUNNING",
                7: "REMOTECTRL",
                8: "ERROR",
                9: "ABORTING',
                10: "CLEARING"
            }

        self.OBCstate = {"state": "UNDEFINED"}

        self.getObjectControlStateClient = self.createClient(GetObjectControlState,
            "/atos/get_object_control_state")
        self.stateReq = GetObjectControlState.Request()
        self.getObjectControlStateTimer = self.create_timer(0.5, self.getObjectControlStateCallback)

    def getObjectControlStateCallback(self):
        while not self.getObjectControlStateClient.wait_for_service(timeout_sec = 1.0):
            self.get_logger().info('service not available, waiting again...')

        future = self.getObjectControlStateClient.call_async(self.stateReq)
        future.add_done_callback(lambda future: self.getObjectControlStateCallbackDone(future))

    def getObjectControlStateCallbackDone(self, future):
        try:
            response = future.result()
        except Exception as e:
            self.get_logger().info('Service call failed')
        else:
            self.OBCstate["state"] = OBCSTATES[response.state]
```

B.4 Drone positioning

```
def getNewDroneOrigin(coordslist, droneOrigin) -> tuple[Coordinate,int]:

    max_x = [0,0]
    min_x = [0,0]
    max_y = [0,0]
    min_y = [0,0]
    x_sum = 0
    y_sum = 0
    coordinateCounter = 0
    for coordList in coordslist.values():
        for coord in coordList:
            y = coord.lat
            x = coord.lng
            x_sum += x
            y_sum += y
            coordinateCounter += 1

            if x > max_x[0]:
                max_x = [x,y]
            if y > max_y[1]:
```

```
            max_y = [x,y]
        if x < min_x[0]:
            min_x = [x,y]
        if y < min_y[1]:
            min_y = [x,y]

x_travel = max_x[0]-min_x[0]
y_travel = max_y[1]-min_y[1]
x_distance = np.sqrt((x_travel)**2+(max_x[1]-min_x[1])**2)
y_distance = np.sqrt((y_travel)**2+(max_y[1]-min_y[1])**2)

if coordinateCounter != 0:
    x_origin_cartesian = x_sum / coordinateCounter
    y_origin_cartesian = y_sum / coordinateCounter

# Earth radius in meters
earth_radius = 6371000

# Convert Cartesian displacements to angular displacements
delta_latitude = x_origin_cartesian / earth_radius * (180 / np.pi)
delta_longitude = y_origin_cartesian / (earth_radius * np.cos(droneOrigin.lat * np.pi / 180))
* (180 / np.pi)

# Calculate new coordinates
new_latitude = droneOrigin.lat + delta_latitude
new_longitude = droneOrigin.lng + delta_longitude
if x_travel >= y_travel:
    deltaY = np.abs(max_x[1]-min_x[1])
    angle = np.arctan2(deltaY,x_travel)
    angle = (angle - np.pi/2)*(180/np.pi)
else:
    deltaX = np.abs(max_y[0]-min_y[0])
    angle = np.arctan2(deltaX,y_travel)
    angle = (angle)*(180/np.pi)

x_distance_with_margin = x_distance + 4
y_distance_with_margin = y_distance + 4

area_to_be_covered = x_distance_with_margin * y_distance_with_margin
height = calculateHeight(area_to_be_covered)
flyTo = Coordinate(new_latitude,new_longitude,height)
angle = round(angle)

return flyTo, angle
```

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sverige 2024
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY