TRACE

inTegration & haRmonizAtion of logistiCs opErations

D4.1 Infrastructural Elements of the TRACE Platform - A

Horizon Innovation Actions | Project No. 101104278

Call HORIZON-CL5-2022-D6-02

Co-funded by the European Union

Dissemination level	Public (PU)
Type of deliverable	R – Document, report
Work package	WP4 – Infrastructure & Ecosystem
Status - version, date	Final v1.1, 08/12/2025
Deliverable leader	National and Kapodistrian University of Athens (NKUA)
Contractual date of delivery	30/11/2024
Actual date of delivery	08/01/2025

List of authors

Author Name	Organization
Vassilis Papataxiarhis	NKUA
Anestis Papakotoulas	NKUA
Sarantis Paskalis	NKUA
Stathes Hadjiefthymiades	NKUA
Sareh Saeedi	CSEM
Panagiotis Kanellopoulos	ACS
Savvas Apostolidis	CERTH
Damian Vizár	CSEM
Martin Sénéclauze	CSEM
Marco Pesci	DIFLY
Alexandros Dalkalitsis	HT
Enrico Pavesi	OLV
Kristijan Percic	PS

Christian Chavez	ROB
Sheila Sánchez	ROB
Sham Puthiya Parambath	UGLA
Alessio Masola	UNIMORE
Paolo Burgio	UNIMORE
Kostas Kolomvatsos	UTH
Panagiotis Fountas	UTH
Christos Kylafas	UTH
Nikolaos Tymplalexis	UNIS
Peggy Papadopoulou	UNIS
Kostas Kolomvatsos	UTH
Panagiotis Fountas	UTH
Christos Kylafas	UTH
Nikolaos Tymplalexis	UNIS
Peggy Papadopoulou	UNIS

Version History

Version	Date	Author	Description of changes
V0.1	15.09.2024	Vassilis Papataxiarhis (NKUA), Stathes Hadjiefthymiades (NKUA), Sarantis Paskalis (NKUA)	Table of Contents specified
V0.2	07.10.2024	Vassilis Papataxiarhis (NKUA)	First round of inputs requested from partners
V0.3	22.10.2024	Vassilis Papataxiarhis (NKUA)	First round of inputs collected and integrated

Version	Date	Author	Description of changes
V0.4	25.10.2024	Vassilis Papataxiarhis (NKUA), Stathes Hadjiefthymiades (NKUA), Sarantis Paskalis (NKUA)	Second round of inputs requested from partners
V0.5	9.11.2024	Vassilis Papataxiarhis (NKUA)	Third round of inputs collected and integrated
V0.6	9.12.2024	Vassilis Papataxiarhis (NKUA)	Text homogenization
V0.7	16.12.2024	Vassilis Papataxiarhis (NKUA)	Version for internal review
V0.8	28.12.2024	Vassilis Papataxiarhis (NKUA)	Comments addressed
V1.0	8.1.2025	Sarantis Paskalis (NKUA)	Submission
V1.1	28.11.2025	Vassilis Papataxiarhis (NKUA), Anestis Papakotoulas (NKUA)	PO Comments addressed

Peer Review

	Reviewer Name	Organization	Date
V0.8	Kostas Kolomvatsos	UTH	23.12.2024
V0.8	Christos Anagnostopoulos	UGLA	23.12.2024
V1.1	Kostas Kolomvatsos	UTH	29.11.2025

Quality Manager Review

	Reviewer Name	Organization	Date
V1.0	Ioannis Neokosmidis	INCITES	8.1.2025

Legal Disclaimer

Executive Summary

This deliverable presents the foundational infrastructure and resource management elements underpinning the TRACE platform for advanced logistics operations. It details the sensing infrastructure—which includes various vehicles (e.g., cargo bikes, autonomous ground vehicles, manned vehicles, drones) and additional sensors—emphasizing how each component captures real-time data to optimize route planning, safety, and operational efficiency.

Moreover, it describes the communication interfaces and networking approaches employed to ensure reliable data exchange. These range from vehicle-to-vehicle (V2V) modules and low-power Time-Division Multiple Access (TDMA) protocols to vehicle-to-infrastructure (V2I) connections leveraging public 4G/5G networks. The deliverable highlights the interplay between on-board data storage, resource management strategies, and backend systems for analytics, ensuring seamless information flow even under variable connectivity conditions.

Key modules—such as the Bike Connection Box, Smart Bike Autonomous Driving System, and the RB-VOGUI Interface—demonstrate how data is gathered, processed, and transmitted for real-time decision-making. The Optimisation Module further refines computational load distribution and task offloading across the cloud-edge-vehicle continuum, keeping resource usage efficient and sustaining autonomy.

In addition to the above, this deliverable incorporates the alignment of the infrastructure with the WP3 system architecture, the application of international standards, and the integration of external data sources. The components described in D4.1 follow the four-tier TRACE architecture and adhere to standards such as ISO/TC 154, ISO 8000, 3GPP, ETSI, ISO/IEC 27001, and ISO/IEC 27701, ensuring secure and interoperable data exchange. The platform also integrates real-time environmental inputs through the Weather Management Systems, as well as traffic information from external APIs. These additions enhance TRACE's interoperability and support more accurate, context-aware decision-making across logistics operations.

Finally, the document describes how these infrastructural components integrate with the TRACE platform's interoperability layer, enabling consistent alignment with the overarching data model and supporting advanced features like real-time event detection and route scheduling. Through this coordinated architecture, TRACE can adapt dynamically to the evolving logistics landscape, ensuring safety, reliability, and efficiency in multi-modal transport scenarios.

Table of Contents

Executive Summary 6

Table of Contents 7

Figures 10

Tables 12

Definitions, Acronyms, Abbreviations 13

1 Introduction 14

1.1 Scope of deliverable 14

1.2 Relation with other work packages/deliverables 14

1.3 Alignment of TRACE Platform Infrastructural Elements with the System Architecture and Integration Strategy 15

1.4 Intended audience 16

1.5 Deliverable structure 17

2 Sensing Infrastructure 18

2.1 Overview of the TRACE sensing infrastructure and Resource Management 18

2.2 Transportation vehicles 19

2.2.1 SUM-X MY 2024 Cargo Bike 19

2.2.2 Robotnik Automation unmanned ground vehicle model RB-VOGUI 25

2.2.3 DIFLY Xplora2 UAS 27

2.2.4 Twinswheel autonomous ground vehicle 28

2.2.5 Cargo trains and supportive systems 31

2.2.6 Cargo trucks and supportive systems 33

2.3 Additional Sensors 36

2.3.1 Low-Power 3D Scene Reconstruction System for Enhanced Platoon Driver Awareness 36

2.3.2 DIFLY LifeBox 38

2.4 Resource management 40
2.4.1 Bike Connection Box 40
2.4.2 Smart Bike Autonomous Driving System 41
2.4.3 Optimisation Module 44
3 Infrastructural Interfaces 47
3.1 Overview of the TRACE infrastructural interfaces and communications 47
3.2 V2V low power communication System 47
3.2.1 V2V reference architecture 48
3.2.2 V2V block diagram 48
3.2.3 V2V communication Security 50
3.2.4 V2V UART protocols 51
3.2.5 V2V TDMA protocols 52
3.3 TRACE, MASA and V2I connectivity 53
3.4 RB-VOGUI Interface 54
3.5 StreamHandler Data Management 58
3.6 Storage and management of local data (on-board) 59
3.7 Network infrastructure provisioning 60
3.7.1 Communication Infrastructure Layer for Italian Demonstrator 60
3.7.2 Communication Infrastructure Layer for Slovenian Demonstrator 61
3.7.3 Communication Infrastructure Layer for Greek Demonstrator 62
3.8 Connection with the several infrastructure data sources used in the pilots 65
3.9 Integrate Real-Time Information and Intelligent Re-Routing Mechanisms 67
3.9.1 Traffic Management System (TMS) 67
3.9.2 Weather Management System (WMS) 68
3.10 Standards and Data Interoperability Framework 72

3.10.1 Sensor Layer 72
3.10.2 Communication Standards 72
3.10.3 API - Data Exchange and Messaging Framework 73
3.10.4 Data ingestion through Excel 73
3.10.5 Data Privacy 73
3.10.6 Weather Data Integration 74
3.10.7 Traffic Data Integration 74
4 Interfacing and Integration with other TRACE Modules 75
4.1 Alignment with the TRACE Data Model 75
4.1.1 Vehicles and the Vehicle Class 75
4.1.2 Real-Time Monitoring via RealTimeVehicleInfo 76
4.1.3 Infrastructure, Areas, and Waypoints 77
4.1.4 Shipments and Goods 77
4.1.5 Load Management: The Load Class 78
4.1.6 Events and Operational Disruptions 79
4.1.7 Capturing Routes with Journey and TransportationMode 79
4.1.8 Populating the Model with Data 80
4.2 Interaction between sensing modules, communication infrastructure, and data processing components 80
5 Conclusion 82
ANNEX A: Python Implementation of Integration Weather Management System 83

Figures

Figure 1: TRACE Platform Conceptual Architecture 15
Figure 2: SUM-X cargo bike parts 19
Figure 3: SUM-X cargo bike dimensions 20
Figure 4: SUM-X cargo bike 20
Figure 5: SUM-X MY2024 E-Bike motor 21
Figure 6: SUM-X MY2024 servo motors 22
Figure 7: 15Ah/36v Battery Pack 23
Figure 8: Cargo Box: Sketch and Dimensions 24
Figure 9: SUM-X equipped with 2m3 cargo box in daily operations 24
Figure 10: Certified 2D safety sensor 26
Figure 11: RB-VOGUI autonomous mobile robot 26
Figure 12: Xplora2 UAS 28
Figure 13: Twinswheel autonomous ground vehicle 29
Figure 14: Cargo trains 32
Figure 15: Electric locomotive Hellas Sprinter 120 32
Figure 16: ACS utilized semi-trailer truck at ACS Athens Central hub 34
Figure 17: ACS' utilized semitrailer truck loaded with roller cages containing shipments 35
Figure 18: Smart vision system with edge processing, allowing for decision making with minimum latency 37
Figure 19: Approaches for 3D reconstruction optimization 38
Figure 20: DIFLY Lifebox 39
Figure 21: Smart Autonomous Driving System modules diagram 42
Figure 22: Example of an Aruco Tag markers for the Camera Leader Detection module 43
Figure 23: Example of an Aruco Tag markers for the Camera Leader Detection module 43
Figure 24: V2V block diagram and interactions 48
Figure 25: V2V firmware upgrade using V2I communication 49
Figure 26: V2V firmware upgrade using V2V communication 49
Figure 27: Connectivity diagram between Entities, Masa, Trace and other application 54
Figure 28: Communication Infrastructure Layer for Slovenian Demonstrator 61
Figure 29: Communication Infrastructure Layer for Greek Demonstrator 62

Figure 30: 4G/5G NSA testbed deployment 64
Figure 31: 5G SA testbed deployment 64

Tables

Table 1: SUM-X my 2024 “TRACE” Power utilization and Range 22
Table 2: Cargo Box Dimension 23
Table 3: Cargo bikes for Last Mile Logistic comparison: Weight, Payload, Capacity 25
Table 4: RSWT Scale Ranges 69

Definitions, Acronyms, Abbreviations

Abbreviation	Definition
ADR	Autonomous Delivery Robot
AWS	Amazon Web Services
BVLOS	Beyond Visual Line of Sight
ECDSA	Elliptic Curve Digital Signature Algorithm
EC	European Commission
FPV	First-person View
GA	Grant Agreement
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HDMI	High-Definition Multimedia Interface
IMU	Inertial Measurement Unit
IoT	Internet of Things
LiDAR	Light Detection and Ranging
MASA	Modena Automotive Smart Area
MQTT	Message Queuing Telemetry Transport
PLC	Programmable Logic Controller
RB-VOGUI	Robotnik Automation unmanned ground vehicle model RB-VOGUI
RGBD	Red, Green, Blue, Depth (camera/sensor)
TDMA	Time Division Multiple Access
TF-M	Trusted Firmware-M (for secure boot)
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
USB	Universal Serial Bus
TRACE	Integration and Harmonization of Logistics Operations
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
WP	Work Package

1 Introduction

1.1 Scope of deliverable

Deliverable D4.1, entitled *"Infrastructural Elements of the TRACE Platform (A)"*, has been prepared in the framework of WP4 *Infrastructure & Ecosystem*, and specifically addresses the design, integration, and specification of the core infrastructural components underpinning the TRACE solution. As the first of two releases (with D4.2 serving as a follow-up deliverable in M36), D4.1 focuses on the technological building blocks (hardware and software) that make real-time sensing and multimodal logistics operations possible within the TRACE environment. In particular, it presents the foundational elements, from vehicle sensors and network architectures to data-collection mechanisms and resource optimization modules, ensuring that all relevant hardware and software interoperate effectively.

1.2 Relation with other work packages/deliverables

D4.1 draws heavily on the functional and technical requirements derived from WP2 *Conceptual Framework*. It builds on the platform-level specifications in WP3 *Platform Design and Integration*, especially T3.1, which covers platform's architecture. By introducing the infrastructural building blocks, this deliverable provides the basis for subsequent tasks:

- Task 3.5 – Platform Integration, detailing the platform integration activities,

- Task 3.6 – Platform Testing and Validation, specifying tests at integration and system level, and

Moreover, D4.1 establishes a foundation for WP5 *Opportunities and Innovation*, particularly in how new logistics business models can capitalize on the integrated infrastructure. Links to WP6 *Large-Scale Demonstration Activities* also exist, as the operational and technical setups detailed here will be validated in real demonstration sites across Italy, Slovenia, and Greece. Finally, D4.1 will feed D4.2, which will update and refine these infrastructural concepts based on ongoing development and pilot feedback.

1.3 Alignment of TRACE Platform Infrastructural Elements with the System Architecture and Integration Strategy

The TRACE Conceptual Architecture prepared in WP3 (Figure 1) outlines how the platform is structured. It breaks the system down into four main tiers: the *User Interaction Layer*, the *Application Services Layer*, the *Data Management Layer*, and the *Infrastructure Layer*. This structure effectively acts as the foundation for the work carried out in WP4 and for the infrastructure described in D4.1. All of the project's hardware, software, and integration elements were designed with these tiers in mind, so the overall system remains consistent from planning through to deployment.

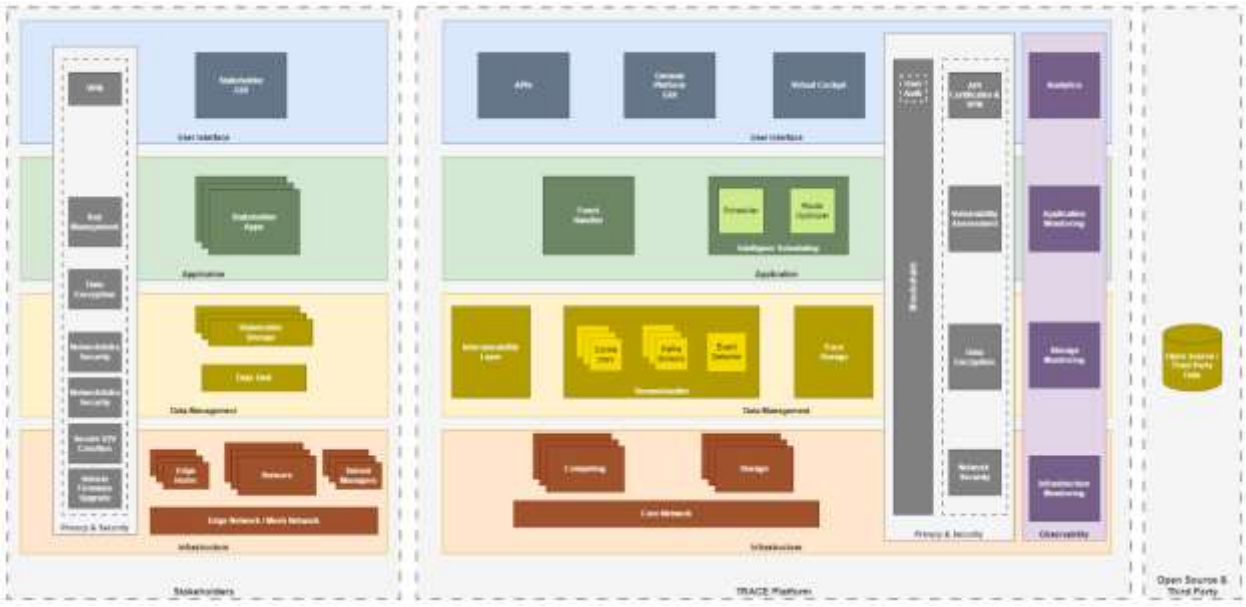

Figure 1: TRACE Platform Conceptual Architecture

In terms of hardware, the infrastructure follows the principles set out in the WP3 Infrastructure Layer and adopts a cloud–edge hybrid approach. The cloud hosts TRACE's core services—such as the Scheduler, Route Optimization, Blockchain, and the Interoperability Layer—while edge nodes located closer to logistics operations handle on-site data capture and early preprocessing. D4.1 provides a detailed breakdown of this setup, which supports real-time reactions, scalability as the system grows, and redundancy where needed. It also ensures that data can move securely and reliably between the edge and the cloud, something that is essential both for the pilots and for day-to-day operation.

On the software side, the Application Services and Data Management layers described in WP3 are delivered through a modular architecture. This includes the Interoperability Layer, the StreamHandler, and the Cloud-based Data Management System (CDMS). Together, these components take care of collecting, harmonizing, and exchanging data across TRACE modules, using standardized RESTful APIs to stay aligned with the integration principles of WP4. The Interoperability Layer plays a central role here, acting as middleware that brings together different types of data sources using the TRACE Data Model and ontology. Additional services—privacy, security, and observability—are embedded throughout the system as cross-cutting modules, ensuring that the platform operates in a secure and accountable way. These are also detailed in D4.1.

The interfaces defined in D4.1 map directly to the Integration Points (IP01 and IP02) set out in WP4 and facilitate communication between the Interoperability Layer, the Scheduler, and the Route Optimization module. They all follow the same API conventions, authentication methods, and data formats, which helps maintain consistency for both internal components and external stakeholders. The architecture also supports connections to outside data sources through dedicated connectors, enabling inputs such as weather data, traffic updates, or transport network statuses. These external feeds are processed through the Interoperability Layer and then used by the scheduling and optimization components to improve predictions and operational robustness.

Overall, the infrastructure described in D4.1 is a direct realization of the conceptual architecture defined in WP3 and follows the integration and deployment framework of WP4. By combining hybrid hardware, modular software, standardized interfaces, and the ability to incorporate external data, TRACE delivers a unified, scalable, and interoperable environment. The alignment across these work packages shows that the system architecture and integration strategy have been fully respected, addressing the reviewer's concerns and meeting the objectives of both WP3 and WP4.

1.4 Intended audience

This deliverable is aimed primarily at technical stakeholders, including system architects, software developers, and integration specialists responsible for setting up, refining, and maintaining the TRACE infrastructure. At the same time, it has been written to be accessible to the broader community—logistics service providers, research institutions, policy-makers, and other external stakeholders—who

require an overview of how the various vehicles, sensors, communication channels, and data-management systems fit together. The dissemination level is *Public (PU)*, making this document available to anyone interested in the project's technical backbone.

1.5 Deliverable structure

The remainder of this document is organized as follows:

- **Section 2** provides an overview of the TRACE sensing infrastructure, detailing the vehicles, sensors, and resource management approach.

- **Section 3** covers the infrastructural interfaces and communication mechanisms, including both vehicle-to-infrastructure (V2I) and vehicle-to-vehicle (V2V) systems.

- **Section 4** outlines the integration of these infrastructure components with other TRACE modules, highlighting alignment with the TRACE Data Model and demonstrating how real-time data is exchanged and processed.

- **Section 5** presents the overall conclusions, summarizing key achievements, challenges encountered, and next steps for further refinement and validation of the infrastructural elements in real-world demonstration scenarios.

2 Sensing Infrastructure

2.1 Overview of the TRACE sensing infrastructure and Resource Management

This chapter discusses the sensing infrastructure of TRACE which is designed to seamlessly collect, process, and transmit essential data related to vehicle performance, cargo conditions, and environmental factors. By incorporating a diverse set of platforms, ranging from cargo bikes and autonomous ground vehicles to trains and drones, the project ensures complete coverage of multimodal logistics operations. This wide spectrum of sensing elements facilitates accurate monitoring of critical parameters, such as real-time location, speed, payload, battery or fuel levels, and ambient conditions, enabling highly informed decision-making throughout the logistics chain.

At the core of the infrastructure, there is a robust resource management framework. TRACE leverages advanced components that dynamically balance tasks among vehicles, edge nodes, and cloud resources, ensuring efficient energy usage and timely data processing under varying connectivity conditions. The system supports, among others, “Follow-me” or platooning scenarios, the offloading of computation-heavy tasks to nearby edge nodes when necessary, and simultaneously accommodates bandwidth-intensive operations such as video streaming and route replanning. In doing so, TRACE can maintain continuous, data-driven operations across diverse environments.

Additionally, the platform’s flexible architecture ensures interoperability among different sensor technologies and vehicle types. Standardized interfaces and communication protocols—like the Bike Connection Box, RB-VOGUI Interface, and specialized drone modules—provide straightforward integration pathways for existing and future mobility solutions. These unified interfaces underpin a scalable approach that can adapt seamlessly to evolving project requirements and logistical demands.

2.2 Transportation vehicles

2.2.1 SUM-X MY 2024 Cargo Bike

The TRACE project allows to create the SUM-X[1] that is an innovative cargo bike that combines lightness and huge load capability. This SUM-X will be used in the TRACE project to demonstrate the Italian Use Case. In the following drawings you can see all its parts:

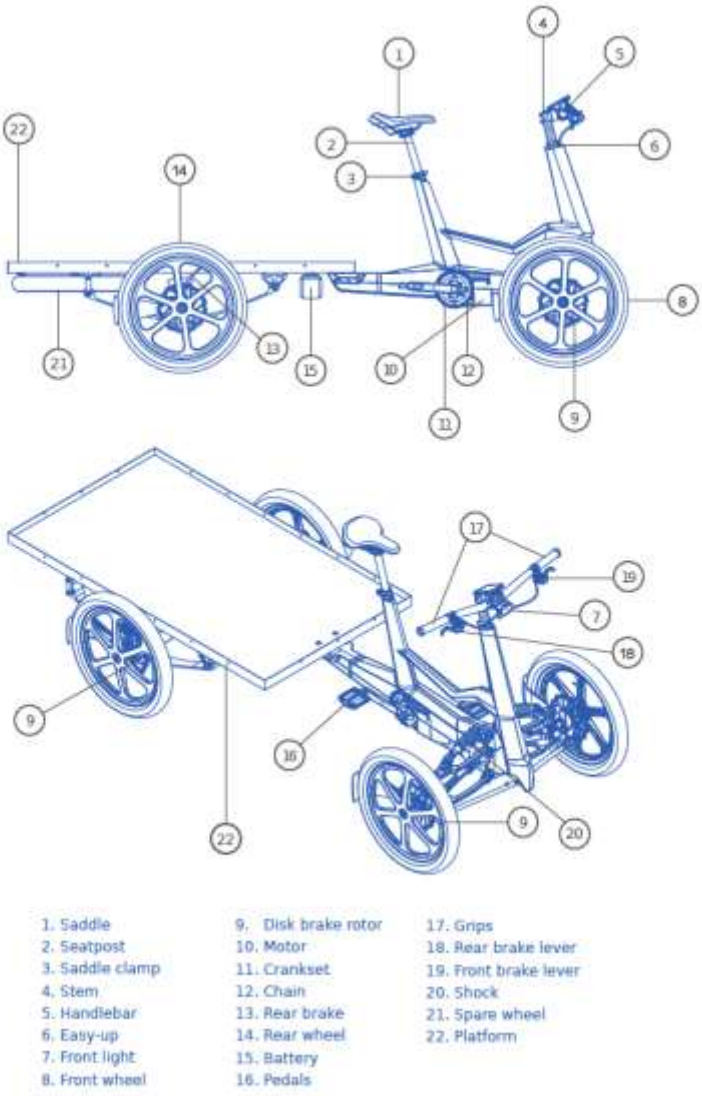

Figure 2: SUM-X cargo bike parts

[1] https://www.sumsolutions.it/en/

Following the drawings with the overall dimensions:

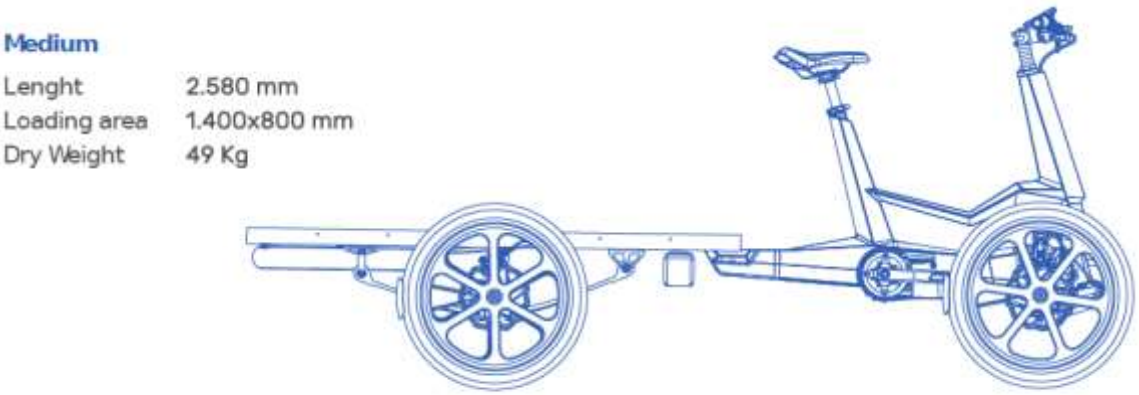

Figure 3: SUM-X cargo bike dimensions

The vehicle has a wheelbase of 970 mm and a steering radius of 4 meters.

Figure 4: SUM-X cargo bike

OLV will produce a custom vehicle for the TRACE project that could be operated both manned and unmanned.

The TRACE version of the SUM-X MY2024 will have a standard E-Bike motor in the front (OLI Sport[2]) coupled with a differential and joint axles that will be used when the vehicle is driven by a human rider, in this mode the vehicle can run up to 25 km/h with a maximum load of 400 kg (rider included).

In order to operate unmanned, a second motor mounted on the rear axle and coupled with a gearbox and a differential will be used to move the vehicle in unmanned scenarios, in this mode the vehicle will be limited to a maximum speed of 6 km/h and will be able to move both forward and backward. Furthermore, this second motor will also act as a lock when the bike is parked and turned off. These features are very appreciated in the last mile logistics, because the riders can't waste time in locking the bike.

The second motor could be set also to operate as a regenerative braking mechanism, from our calculation, if properly set, this can save up to 5% of energy during delivery operations.

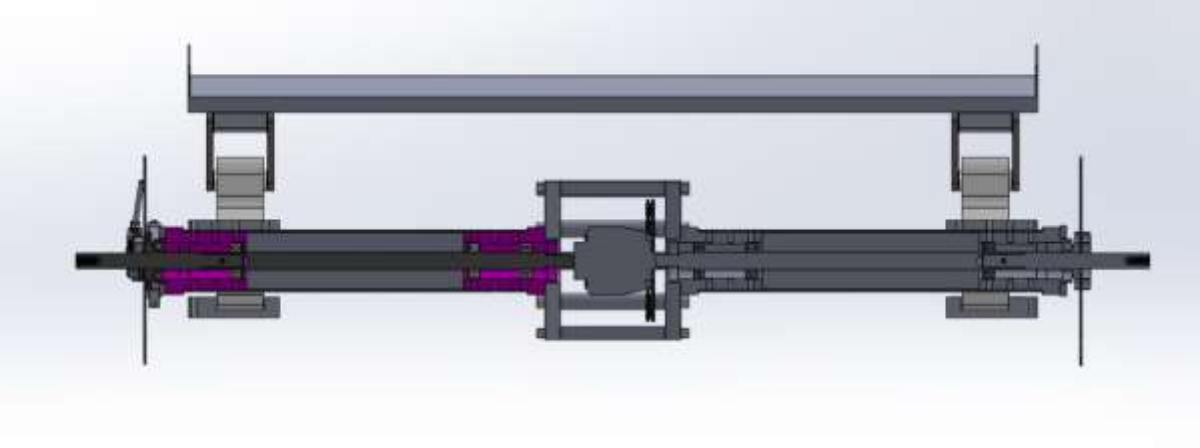

Figure 5: SUM-X MY2024 E-Bike motor

Two other servo motors will be integrated on the vehicle:

1. one to operate the steering mechanism, placed on the lower part of the steering tube, and

[2] https://www.oli-ebike.com/

2. one to control an additional brake system on the rear axle that will be used for both parking and safety reason.

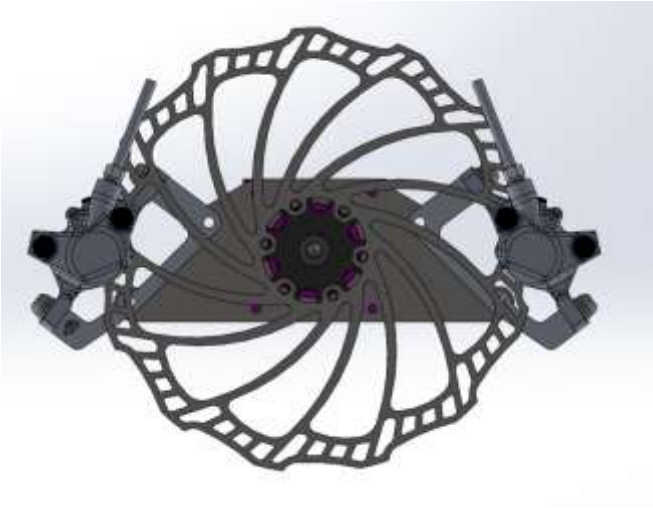

Figure 6: SUM-X MY2024 servo motors

All the devices that allow to control remotely the vehicle are shut off when it is operated by a human rider.

Table 1: SUM-X my 2024 “TRACE” Power utilization and Range

Utilization Mode	Max Speed	Total Mass, kg *(Vehicle + Rider + Max payload)*	Ev. Power Utilization, W/km *(Flat city)*	Max Range, km
Manned	25 km/h	400	17	31
Unmanned	6 km/h	300	12	45

Max Riders weight considered 100kg, power utilization is significantly reduced due to lower running speed and reduced total mass (no rider).

The vehicle will be equipped with a 15Ah Battery 36v. For extended range is possible to add an additional battery (available battery slot is already implemented in the bottom part of the carbon flat bed) or a larger battery pack among standard available solutions (17Ah).

Figure 7: 15Ah/36v Battery Pack

The vehicle is equipped with a single-compartment cargo box, equipped with a rear door locking system. The dimensions of the BOX are shown in the table below.

Table 2: Cargo Box Dimension

Width	886 mm
Length	1428 mm
Height	1539 mm
Total Volume	1497 L

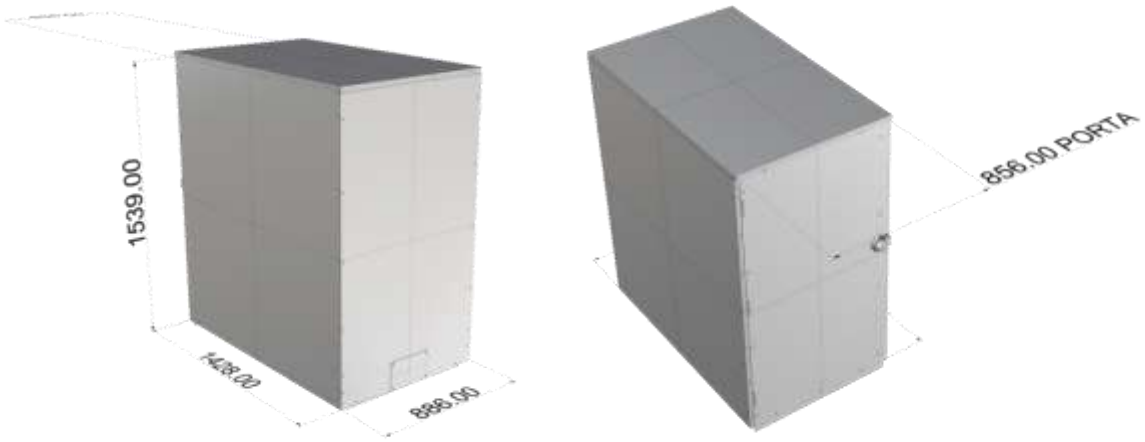

Figure 8: Cargo Box: Sketch and Dimensions

Figure 9: SUM-X equipped with 2m3 cargo box in daily operations

The cargo box is large enough to keep a carrier running for the duration of a shift. In daily operations the box is fully loaded, the volume and quantity of parcels shipped is comparable to traditional motor Van.

Table 3: Cargo bikes for Last Mile Logistic comparison: Weight, Payload, Capacity

				
Vehicle Name	EAV	Cargo Cycling	Rytle	SUM-X Trace Version
Dry Weight (kg)	125	190	140	75
Max Payload (kg/m3)	150 /1.6	200 / 1.5	160	250 / 1.9

2.2.2 Robotnik Automation unmanned ground vehicle model RB-VOGUI

The RB-VOGUI is a highly adaptable mobile robotic platform primarily designed for outdoor transport tasks. This versatile solution features an all-terrain, omnidrive kinematic base that enables seamless mobility in both indoor and outdoor environments, even across complex terrains. Its design makes it suitable for automating various tasks in industries such as agriculture, construction, logistics, and R&D, with potential applications like last-mile delivery.

Key features include autonomous navigation capabilities, allowing the robot to localize, map its environment, and perceive its surroundings without human intervention. With its dimensions of 1,250x800x1,300 mm (length, width, height), the platform can achieve a maximum speed of 2.5 m/s, making it efficient for a wide range of operations. It boasts a payload capacity of up to 150 kg, with a maximum compartment size of 780x580x600 mm. Battery life under typical usage conditions extends to 4 hours, with a maximum of 9 hours when inactive. Full recharging takes approximately 4 hours.

The robot is equipped with an array of advanced sensors for environmental recognition and interaction, including IMU, GPS, and a 3D LiDAR system. Safety is ensured by two certified 2D outdoor safety scanners, a certified safety PLC, and traditional sensors such as an RGBD camera. These allow the RB-VOGUI to effectively navigate, detect obstacles, and ensure safe operation in both urban and rural settings.

Figure 10: Certified 2D safety sensor

The RB-VOGUI can operate in a range of environmental conditions, from -10°C to +45°C, over distances of up to 14 km, depending on specific conditions. It can handle a maximum slope of 47% and climb steps of up to 30 mm, showcasing its robustness on uneven terrains. With an IP53 waterproof rating, the platform is well-suited for outdoor use. Communication capabilities include Wi-Fi and Bluetooth modules, alongside connectors for USB, RJ45, and HDMI, providing flexibility in terms of external communication and data exchange.

Figure 11: RB-VOGUI autonomous mobile robot

The RB-VOGUI robot is optimal for last-mile delivery in outdoor environments, particularly within the context of the TRACE project, due to its capability to navigate challenging terrains and its autonomous navigation equipped with advanced sensors. Furthermore, its safety sensors are certified and accompanied by a safety PLC that cuts off the robot's power supply in the event of a collision risk,

thereby ensuring pedestrian safety. Its robust design, with an IP53 rating, a payload capacity of 150 kg, and a battery life of 4 hours, facilitates effective integration into complex urban environments.

2.2.3 DIFLY Xplora2 UAS

This drone is a battery-powered aerial platform designed for the fast and efficient transport of light payloads. Weighing 20 kg, it can carry up to 4 kg of cargo and reach a maximum speed of 10 m/s. The battery lasts for 35 minutes under full load and can extend up to 50 minutes without a load, making it ideal for operations that require speed and flexibility.

The drone is equipped with constant GPS tracking, allowing real-time monitoring during flights to ensure secure and reliable operation. It can accommodate a maximum load width of 40 cm, offering versatility in the types of materials it can transport.

With its advanced features and robust structure, the drone serves as a versatile and high-performance solution for transport operations across various settings, from urban deliveries to remote environments.

The drone is equipped with an IMU (Inertial Measurement Unit), GPS, and barometer to maintain stable and precise flight. The GPS, combined with an LTE module, enables accurate position sharing in real-time, enhancing situational awareness and coordination during operations. Additionally, a camera provides FPV (first-person view) images, assisting the pilot in Beyond Visual Line of Sight (BVLOS) operations. These FPV images can also be shared with the TRACE platform or other stakeholders to monitor the overflown area, supporting additional logistics and operational systems.

Reliable GNSS signal and cellular connectivity are necessary infrastructure for correct operations, ensuring accurate geolocation and continuous communication.

Images from overflown areas can be used to improve situational awareness in real time but also to address further services.

Analysis of video streams on the overflown areas can be used to make statistics of actual population in the area, which is an augmentation of the information necessary for flight authorization.

Artificial intelligence algorithms can detect human presence in the field of view and, using geographical information and exact positioning, compute the population density in the overflown areas throughout all flights.

This can be an example of improved service that can be offered through the deployed system, but is not currently foreseen in TRACE platform (data streaming, saving and GDPR handling, analysing data are not addressed).

Figure 12: Xplora2 UAS

2.2.4 Twinswheel[3] autonomous ground vehicle

In the TRACE project, Pošta Slovenije has committed to utilizing one of the commercially available autonomous delivery robots as part of its pilot demonstrations and integration into TRACE platform. The selection was based on key criteria, including accessibility, connectivity, cost-efficiency, functionality, provision of service support, staff training for demonstration activities, and prior experience with other European postal operators. We have chosen the French manufacturer, Twinswheel, as the most suitable provider. Their autonomous delivery vehicle offers a broad range of functionalities, from a basic “follow me” feature to fully autonomous parcel delivery and collection across multiple addresses within a single route.

[3] Twinswheel Droid Manufacturer Corporation, https://www.twinswheel.fr/

The company Twinswheel offers three different sizes of ADRs (City S, M, and L); for testing, we have selected the M model, which will be equipped with multiple separate compartments to ensure secure delivery to end users.

Figure 13: Twinswheel autonomous ground vehicle

Specifications of Twinswheel ADR

- 4 driving, suspended and steering wheels
- 4 driving modes: radio control, remote 4G control, follow-me, autonomous on virtual roads
- LiDAR sensors + 2D & 3D cameras + IMU + encoders + GNSS
- Position, brake, turn signal lighting
- Eyes and sounds
- Wi-Fi and 4G connectivity (excluding SIM card)
- Multi-locker box with touch screen
    - box weight: 50 kg, capacity: 100 kg

- Total number of lockers: 6

- Teleoperation station (computer with 3 screens)

- Dimensions: length 1500 x width 690 x height 1370 mm,

- Weight empty: 150 kg

The performances are:

Maximum speed: 12 km/h

Autonomy: 15 km, exchangeable batteries

With the provided web applications, the robot is able to move autonomously in an urban environment (narrow streets) with a moderate number of people around it.

The Twinswheel autonomous delivery robot enables straightforward terrain mapping through advanced sensors, including 3D cameras, 2D and 3D LiDARs, and other sensor types. The data collected during terrain mapping is used to generate a spatial layout where the ADR can navigate. Pickup location points, linked to recipients' addresses, are entered for precise routing. The ADR is equipped with a large array of sensors that allow for efficient and safe navigation within the mapped area, as well as API connectivity with various systems, such as postal information systems and the TRACE system.

The robot also enables real-time remote monitoring and control takeover in critical situations, corresponding to SAE Level 4 autonomy. Safety is ensured by five different systems.

In the pilot, we will test various usage scenarios, such as:

- picking up parcels at a consolidation center and delivering them to different addresses,
- collecting parcels from multiple customers and delivering them to a consolidation point,
- picking up parcels from a single customer and delivering them to multiple recipients.

Parcel sizes will be limited to compartment sizes, primarily S and M, which represent 80% of Pošta Slovenije's parcel traffic.

Parcel Collection and Delivery Process

Upon receiving a parcel in the PS system (or notification of its receipt), users will be able to select ADR delivery through the "My Delivery, My Choice" feature offered by Pošta Slovenije. Using the TW and TRACE systems, users can specify a time window for pickup at the pilot location. Once the parcel is loaded onto the autonomous delivery vehicle (within the selected time window), the user will be notified that the parcel will be delivered within a specified timeframe of 0.5 to 2 hours. A few minutes before delivery, the system will notify the recipient to proceed to the pickup location outside the building. The recipient will receive an access code via their phone for a specific compartment on the autonomous vehicle. Upon collection, the TRACE system will automatically record the pickup time and location. A similar procedure will be followed for parcel collection.

2.2.5 Cargo trains and supportive systems

The Thriasio – Thessaloniki terminal to terminal service offered by HT involves the transportation of intermodal transport units (containers) along the axis via regular railway routes. The rail transport is carried out using container platform wagons, "Rgss" type and "362" series. The transport takes place with containers 45', 40', 20' normally provided by the customer. Containers can be both pallet wide and high cube. HT receives the sealed container from the customer.

Maximum gross weight per container: 30,00 tn

For large regular customers the service defines customers that can provide at least 2 containers per destination. Regarding the Rgss 362 wagon the specifications according UIC are the following:

Length: 18.520 meters

Tare weight: 23.5 tn

Maximum gross weight: 56 tn

Loading length: 60 '

Number of axis: 4

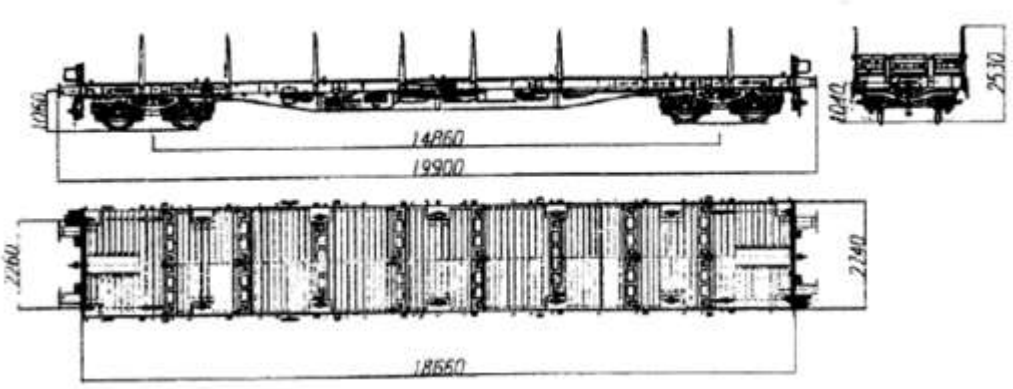

Figure 14: Cargo trains

A train cannot carry more than 1200 gross tn of this type of wagon or 600 meters length of wagons. The main electric engine of a series of wagons in Athens Thessaloniki route is the electric locomotive Hellas Sprinter 120. This engine is 80tn and it has 5.000KW of power.

Figure 15: Electric locomotive Hellas Sprinter 120

2.2.6 Cargo trucks and supportive systems

The trucks utilized by ACS for daily shipments for the corridors from Athens to Thessaloniki and vice versa fare semi-trailer trucks. These vehicles are integral to long-haul freight transport, combining a tractor unit with a semi-trailer. These type of trucks with the specifications provided below will be the ones that will be used for the Greek scenario of the TRACE project.

Below are their typical specifications:

Dimensions (in m):

- Overall Length: Approximately 16.5
- Width: 2.5
- Height: 4
- Trailer Length: Standard trailers are about 13.6m long. s

Capacity:

- Maximum Gross Vehicle Weight (GVW): Up to 40 metric tons, depending on regional regulations.
- Payload Capacity: Approximately 24 to 26 metric tons.
- Volume Capacity: Around 90 m^3.

Engine and Performance:

- Engine Power: Typically between 400 to 600 horsepower.
- Fuel Type: Diesel is standard, though some models may use alternative fuels. ACS utilizes those with the standard fuels at the moment.
- Fuel Consumption: Approximately 35lt/00km.

Roller Cages:

- Capacity per Truck: Each truck carries 36 roller cages.
- Capacity per roller cage: Approximately 110 shipments

Figure 16: ACS utilized semi-trailer truck at ACS Athens Central hub

Roller Cage Specifications follow:

1. Dimensions (in m):
    a. Width: 0.8
    b. Length: 1.1
    c. Height: 2.2
2. Capacity:
    a. Volume: About 1,9 m^3
    b. Weight Capacity: Typically supports up to 500 kilograms
3. Structure:
    a. Material: Generally made from steel with a wire mesh or solid walls for durability and security.

b. Wheels: Equipped with swivel caster wheels, often with brakes, to allow easy manoeuvring and stability during loading/unloading.

4. Security:
    a. Locking Mechanism: Locking bar or latch to secure items during transport.
    b. Label Holders: Often include areas for labelling or barcodes to identify contents.

Figure 17: ACS' utilized semitrailer truck loaded with roller cages containing shipments

2.3 Additional Sensors

2.3.1 Low-Power 3D Scene Reconstruction System for Enhanced Platoon Driver Awareness

The Low-Power 3D Scene Reconstruction System is designed to improve driver awareness in platooning scenarios by providing real-time, three-dimensional reconstructions of the surrounding environment. This system aims to offer significant advantages, particularly in high-speed and complex environments, such as motorcycle platoons, where quick decision-making and object detection are critical for safety. By utilizing a low-cost, low-power setup, the system ensures energy efficiency without compromising performance, making it suitable for use in both individual motorcycles and larger platooning formations. One key benefit is its ability to operate reliably at relatively high speeds, allowing for smooth and continuous feedback to the driver. This enhances situational awareness, enabling better decision-making and minimizing risks in fast-paced, dynamic environments. A key feature of this system is its ability to support the transition from Level 3 to Level 4 autonomy. In Level 3, the system enables platooning in a semi-autonomous “follow-me” mode, allowing vehicles to follow a lead vehicle with minimal driver intervention. The integration of advanced 3D scene reconstruction further enhances this by improving obstacle detection and situational awareness. As the system evolves toward Level 4 autonomy, it will allow fully autonomous operation.

To address privacy concerns, the system leverages edge processing, which processes data locally on the vehicle rather than transmitting it to centralized servers. This method not only reduces latency but also protects sensitive information, such as real-time visual data, from being exposed to external networks. The integration of this 3D reconstruction system with the Smart Bike Autonomous Driving System (developed by UNIMORE) strengthens the overall TRACE platform, creating a cohesive system that merges 3D scene awareness with autonomous bike navigation.

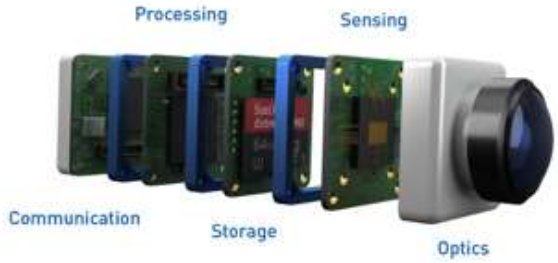

Figure 18: Smart vision system with edge processing, allowing for decision making with minimum latency

We have been exploring various optical layouts to optimize 3D reconstruction, specifically focusing on stereo vision and omniview-based systems. Our primary objective is to reduce computational costs at the edge, making real-time processing feasible. Current 3D reconstruction methods typically start with the calibration of camera intrinsic and extrinsic parameters, followed by stereo matching to identify corresponding points, and conclude with coarse-to-dense reconstruction to obtain a detailed 3D model. This approach is computationally intensive and challenging to implement for real-time applications.

To address these limitations, we are investigating a holistic optical layout in which stereo matches are pre-located at defined positions within the camera's focal plane. This configuration enables a simplified triangulation process, resulting in a linear relationship that bypasses the need for exhaustive search across detector planes. This method significantly reduces processing time, making real-time 3D reconstruction achievable.

This advancement is particularly beneficial for features such as "Follow-me" and dynamic platoon management within the Smart Bike system. Enhanced scene understanding via fast 3D reconstruction supports safer and more efficient coordination in platoon formations. Additionally, real-time 3D scene reconstruction directly enables embedded obstacle avoidance, ensuring the safety of bike riders and those around autonomous systems. Should our holistic approach not meet the required performance standards, a fallback plan utilizing conventional stereo matching and compressive matching pairs will be implemented to ensure robust performance.

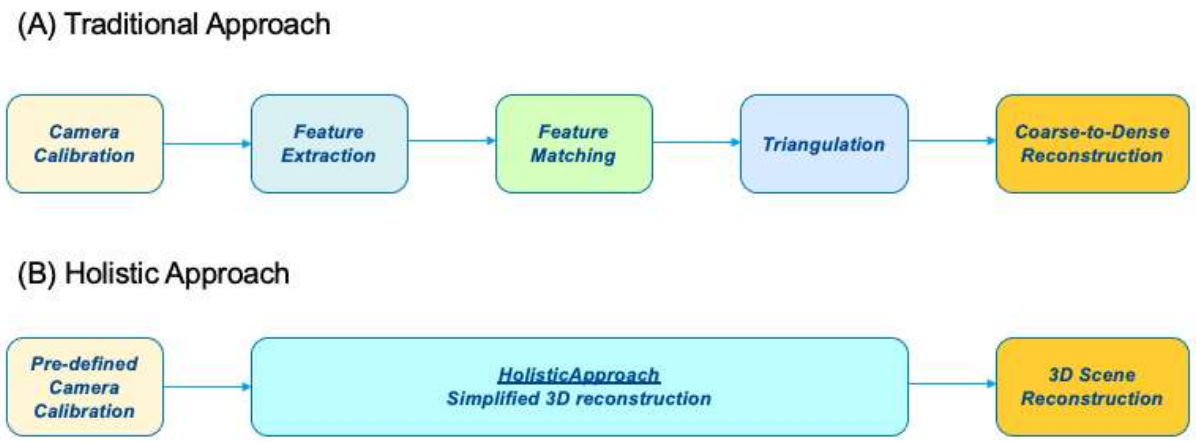

Figure 19: Approaches for *3D reconstruction optimization*

2.3.2 DIFLY LifeBox

The LifeBox is a specialized container designed to support biomedical transport across various phases and environments, including hospitals and drones. It is easy to handle by healthcare personnel, allowing for the insertion and extraction of primary containers in a sterile environment. Additionally, it can be managed by medical or paramedic staff, who can set parameters such as temperature settings, destination, and acceptable limits for shocks, vibrations, and temperature variations.

The LifeBox is versatile and can be transported by multiple means, including by hand, bicycle, drone, and car. It is integrated with a logistics network that provides constant tracking to ensure continuous monitoring and secure delivery of sensitive medical materials.

To ensure optimal conditions, the LifeBox is equipped with various sensors, including a Raspberry module with LTE, GPS for geolocation, an accelerometer to detect vibrations and shocks, two temperature sensors (one internal and one external), and a wattmeter. The external temperature sensor can share data with the TRACE platform, providing environmental information that can support logistics, such as weather conditions.

This advanced sensor setup allows the LifeBox to provide monitored and secure transport conditions, ideal for delivering delicate medical materials across various settings and modes of transportation.

Lifebox is a connected logistics item. Information shared by Lifebox with Modena Automotive Smart Area (MASA)[4] and TRACE platform, provides a redundant information that corresponds to the vehicle that is carrying it (drone or bike). This provides a crosscheck and supports tracking the hand-over phase. Remark that this connected box allows handling hospital logistics beyond the capability of TRACE vehicles, because it also covers the phases that are handled by hospital personnel and with no vehicles, including loading and unloading phases. This enhances the framework and scope of medical logistics.

Figure 20: DIFLY Lifebox

[4] Modena Automotive Smart Area, https://www.automotivesmartarea.it/?lang=en

2.4 Resource management

2.4.1 Bike Connection Box

The Bike Connection Box is a hardware and software system developed by UNIMORE (University of Modena and Reggio Emilia) to facilitate communication between autonomous cargo bikes and the TRACE platform. This system, built on a previous project called City Box, is designed to manage data exchange, monitor operational parameters.

Key Features include:

Connectivity: Ensures reliable communication between autonomous bikes and the TRACE system via 4G/5G.Supports V2V (Vehicle-to-Vehicle) communications to share data such as bike location and delivery status.
Real-time Data Processing: Continuously streams live data to the TRACE platform, including GPS location, parcel status, and delivery route updates passing through Modena Automotive Smart Area (MASA)[5].
The Bike Connection Box facilitates the exchange of real-time sensor data, including:

- Camera frames for situational awareness (optional for privacy reason).
- Vehicle information like battery status, remaining range, and current load in kilograms.
- Altitude data for airborne vehicles.
- Operator presence status.
- Maximum vehicle load capacity and current weight being transported.

This ensures accurate monitoring and management of operational status, improving decision-making for logistics and safety during autonomous deliveries.

Allows also the platoon leader to reorganize the bike platoon when needed based on real-time data.

Integration with TRACE Platform:
Fully integrated with TRACE for seamless routing and parcel management specifically, by using the connectivity with the StreamHandler, that will be used to exchange information with different TRACE's

[5] [5] Modena Automotive Smart Area, https://www.automotivesmartarea.it/?lang=en

modules as Scheduler & Route Optimizer, the Event handler as the Resources Monitoring and Events Manager, Event Management Module, Monitoring Module, Fleet Monitoring Manager, Mitigation Manager and with the Virtual Cockpit. Enables the efficient reallocation of resources, such as sending empty bikes back to the hub or coordinating drone handovers.

2.4.2 Smart Bike Autonomous Driving System

The Smart Bike Autonomous Driving System integrates advanced sensors, communication interfaces, and AI-driven control to enable self-driving capabilities for cargo bikes. Key features include:

- Autonomous navigation in a platooning mode using onboard sensors (cameras, LiDAR).
- Communication with the TRACE platform for real-time route updates.
- "Follow-me" mode, allowing vehicles to follow a leader or other vehicle in a platoon formation.
- Capability to exchange information via the Bike Connection Box (BCB), enabling the transfer of packages between different vehicles.
- Dynamic platoon management and manual override for safety.
- Automated short-distance trips or parking maneuvers can be initiated for battery recharging or when the vehicle is empty, based on real-time optimization decisions from the TRACE platform

This system enhances operational efficiency and ensures seamless coordination in autonomous delivery processes.

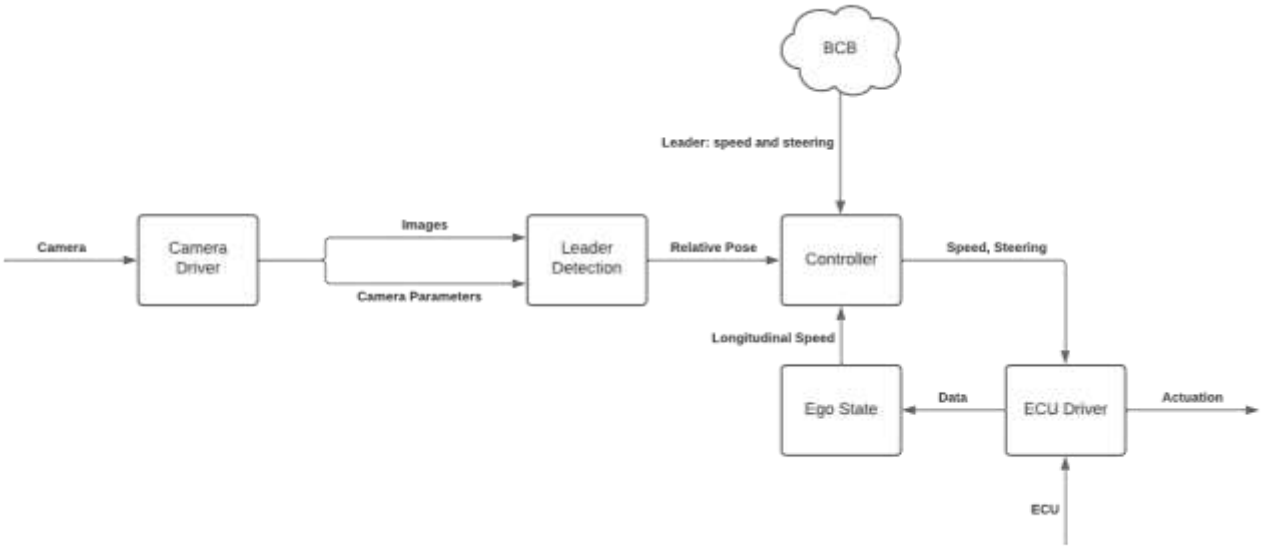

Figure 21: Smart Autonomous Driving System modules diagram

Components:

- Camera Driver: This is a basic software interface that connects to the camera via Ethernet or serial connection. It collects image data and republishes it through ROS2 (Robot Operating System 2), allowing other components to access the camera feed.
- Camera Leader Detection: This component processes the images received from the camera and identifies the next vehicle in the platoon (the "Leader"). Each vehicle is equipped with an Aruco Tag, which is detected to compute its 3D position, orientation, and velocities. The system works in relative space, meaning no global positioning is required.
- Controller: The controller computes movement commands for the vehicle based on the leader's position and speed. It considers the current relative position and speed of the leader, predicting future movements with a safety margin. For lateral movement, the system uses the Simple Bicycle Kinematic Model, adjusting steering based on the target heading and vehicle specs (like wheelbase and steering limits). The desired speed and steering inputs are then sent to the ECU for execution.
- ECU Driver: This low-level software interfaces with the vehicle's ECU (Electronic Control Unit) to collect essential data like current speed, steering angle, and throttle/brake status. It also sends control commands to maneuver the vehicle, connecting higher-level decisions with low-level hardware.
- Ego State: This module processes and republishes the data from the ECU (e.g., speed, steering), enabling higher-level components (like the controller) to use it. It can also perform additional computations to derive other useful metrics. If extended, it could incorporate SLAM (Simultaneous Localization and Mapping) to use LiDAR data for real-time localization in both known and unknown environments.
-

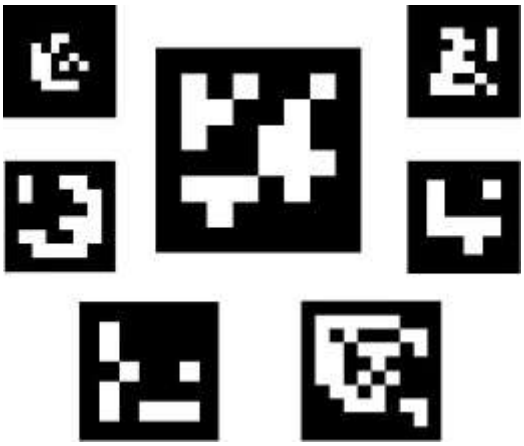

Figure 22: Example of an Aruco Tag markers for the Camera Leader Detection module

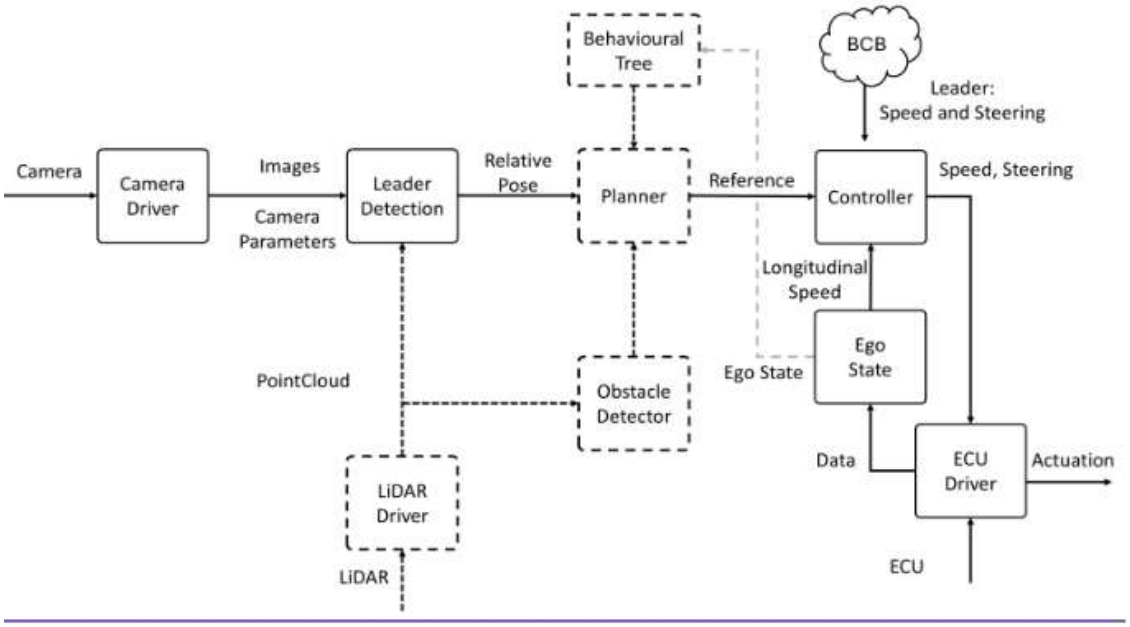

Figure 23: Example of an Aruco Tag markers for the Camera Leader Detection module

Possible future components:

- LiDAR Driver: A simple driver that retrieves data from the LiDAR sensor and republishes the corresponding PointCloud.
- LiDAR Leader Detection: This component receives the PointCloud from the LiDAR driver and extracts the relative position of the leader using classical geometric methods, such as clustering.

- Leader Tracking: Implements a Kalman Filter to combine detections from both the camera and LiDAR, associating them with the same object and tracking the leader. It handles brief periods of lost detection (due to occlusions, adverse weather, etc.) using a simple model like a point mass.
- Obstacle Detector: This component takes the LiDAR PointCloud and detects obstacles through classical geometric methods like clustering. It matches clusters against the camera's leader detections, marking unmatched clusters as obstacles to avoid.
- Planner: Responsible for computing a feasible path that follows the leader while ensuring safe navigation. It avoids obstacles using an RRT* algorithm, which samples the state space and connects the ego position to the leader's relative position. It also defines no-go zones around detected obstacles with safety margins.
- Behavioral Tree: Acts as a supervisor for the entire AD software stack. It can stop or slow down the vehicle in abnormal conditions (e.g., hardware or sensor failures). It also triggers automated behaviors, such as parking or docking, under certain conditions like low battery levels.

2.4.3 Optimisation Module

The TRACE resource management and optimization framework is designed to enhance the utilization of resources and facilitate tasks management in mobile edge computing (MEC) environments in the scope of smart and multimodal transportation systems being active in the logistics domain. It leverages onboard computational resources in transportation vehicles, such as unmanned aerial or ground vehicles and manned vehicles, while dynamically integrating with the more (computationally) advanced MEC infrastructure. We have to notice that the MEC infrastructure acts as the intermediate between the envisioned vehicles and the Cloud back end where a processing adopting more advanced resources can be realized. This 'vertical', i.e., Cloud-Edge-Vehicles creates numerous opportunities for the management of the available resources and tasks according to needs facilitating the optimization of the processing in different layers.

The TRACE resource management framework strongly supports stochastic environments where network conditions, vehicle availability, and computational demands fluctuate. It is designed to handle scenarios such as unexpected shifts in resource availability adapting dynamically to maintain efficiency. The TRACE resource management framework also handles virtualized onboard computational resources enabling the flexible allocation of tasks across vehicles or between the cloud-edge computing continuum. By

virtualizing resources, the system can efficiently manage workload distribution based on the operational requirements of each vehicle and its capabilities. This approach enables interoperability among diverse vehicle types (with varying OS and hardware platforms) and seamless integration with MEC nodes. This way it fosters scalability and adaptability. The onboard processing capabilities and resource management facilitates the real-time decision-making arming vehicles with the necessary intelligence to fulfill their logistics missions and react in the changes of the surrounding dynamic environment.

Another core component is the system's capability to decide dynamically whether tasks should be executed locally on the vehicles or be offloaded to more advanced MEC nodes. Task offloading is guided by contextual information, such as network conditions, battery life, and computational load of the node. For instance, tasks that demand significant computational power can be transferred to nearby MEC nodes to conserve the vehicle’s battery until the delivery of goods to their destination is completed and maintain optimal performance. The local processing system constantly monitors these parameters to make informed decisions about offloading, ensuring, at the same time, elasticity in resource management.

Time-based optimization also plays a critical role in the framework, incorporating concepts from optimal stopping theory. This includes determining whether to transmit large amounts of data, such as video streams, based on network load and resource availability. Optimal stopping intends to balance resource efficiency with the need for timely completion. This approach is applied to scheduling tasks that involve network and computational resources, such as deciding when to aggregate or transmit data to minimize bandwidth usage without compromising the system's responsiveness. It becomes clear that such a functionality arms the vehicles with a proactive ‘reasoning’ mechanism capable of understanding and reacting to potential future changes in the environment as it is understood by the recording of the envisioned parameters.

It is obvious that unmanned vehicles working in an autonomous mode, to fulfill their mission, have to monitor a set of sensors and execute various tasks. This adds burden in the local resources with apparent effects in, e.g., the battery life, etc. TRACE alleges that an optimization scheme should be applied in the management of tasks, i.e., to decide when a task or a set of tasks should be executed/offloaded in the Edge infrastructure to save the local resources and avoid jeopardizing the mission. The continuous recording of sensor values and the status of the vehicles will be combined with the needs of the desired

tasks. We envision a scheme where tasks are placed in a local structure (e.g., a queue, maybe with a pre-emptive/prioritization approach or not) and the local processing unit continuously reason upon their 'safe' execution. With the term 'safe', we denote the uninterrupted and smooth execution of tasks under the current and future status of the resources (e.g., battery status, network connectivity, etc). Tasks have specific requirements like the estimated time for execution, estimated energy demand, the desired processing, the required data, etc and based on these requirements the local processing unit may decide if a task or a number of tasks should be offloaded to the edge infrastructure. For instance, let us imagine that a logistic actor wants to activate the virtual cockpit and see the camera recordings in real time after receiving the indication of an unexpected event while, at the same time, the vehicle executes a processing for calculating a new route affected by the aforementioned event. Let us assume that the remaining resources are not enough to have locally executing both processing activities, the envisioned delivery and the return back to the hub. Then, the vehicle decides which task should be offloaded in the edge infrastructure possibly applying a prioritization mechanism. Naturally, the vehicle may decide to open the camera and send back to the virtual cockpit the feed, i.e., video or images (it depends on the remaining resources and the outcome of the optimization scheme) for the minimum possible time (calculated again by the optimization scheme to avoid jeopardizing the battery life required to complete the mission) while, at the same time, it requests the calculation of the best possible (new) route out of a number of routes from the edge infrastructure. This mechanism is applied upon the virtualized resources placed not only on the vehicle but also on the edge infrastructure and continuously secures that the mission is not jeopardized. The proposed approach is accompanied by the necessary, however, simple modelling of tasks and their requirements and a simple, however, powerful, reasoning mechanism for realizing the offloading activities.

3 Infrastructural Interfaces

3.1 Overview of the TRACE infrastructural interfaces and communications

This chapter introduces the core communication frameworks and interfaces that bind together the diverse vehicle types, sensors, and backend systems in TRACE. By leveraging both low-power local networks (e.g., V2V) and wide-area connectivity (e.g., 4G/5G networks), the TRACE platform achieves robust, real-time data exchanges, enabling continuous monitoring and management of logistics operations. These communication systems are designed with adaptability in mind—capable of supporting both small-scale platooning deployments and large intermodal transport corridors.

A key achievement is the integration of different communication layers (i.e., V2V, V2I, and edge-to-cloud) into a seamless data pipeline. This pipeline ensures that sensor readings and status updates from cargo bikes, ground robots, drones, and trains reach the TRACE platform reliably, regardless of environmental or connectivity constraints. By standardizing protocols and interfaces, the system achieves high interoperability among devices, paving the way for rapid scaling and easy extension to new use cases.

Additionally, the chapter delves into the security and reliability aspects of TRACE's communication infrastructure. From firmware updates over TDMA radio links to authentication mechanisms for public 5G networks, multiple safeguards are in place to protect data integrity and maintain system availability. The result is a resilient architecture that underpins the entire platform, ensuring that real-time insights continuously inform route optimization, event detection, and resource allocation across all TRACE deployments.

3.2 V2V low power communication System

The low-power communication system embedded in each cargo bike will facilitate direct communication among all bikes within the Trace Platoon. While communication could be managed through the Vehicle-to-Infrastructure (V2I) system, having a local, infrastructure-independent communication method significantly enhances both robustness and security. This local system ensures that, even if 5G connectivity is lost, the platoon can continue to operate autonomously and cohesively. In addition, this

V2V communication system will be used for firmware update when bike will be back in their storage location.

3.2.1 V2V reference architecture

The V2V communication component is composed of a CPU and a communication module. The module has been built to allow simple swap of this communication module based on the needs. For the trace project, the module will allow 2.4Ghz communication using TDMA based protocol.

3.2.2 V2V block diagram

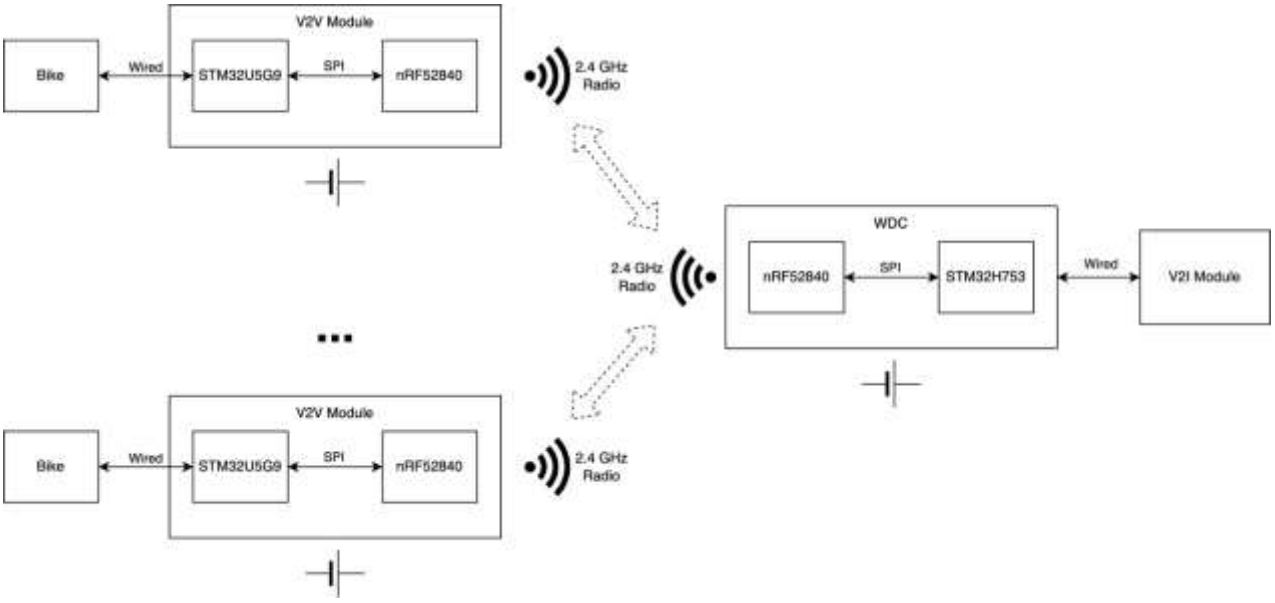

Figure 24: V2V block diagram and interactions

It is connected to the V2I module of each cargo bike through a serial port. In its first version, the tandem Serial-5G will be used transparently for the firmware update of the V2V. The next version should allow the upgrade to be done directly through the V2V communication based on the version hosted on the main cargo bike.

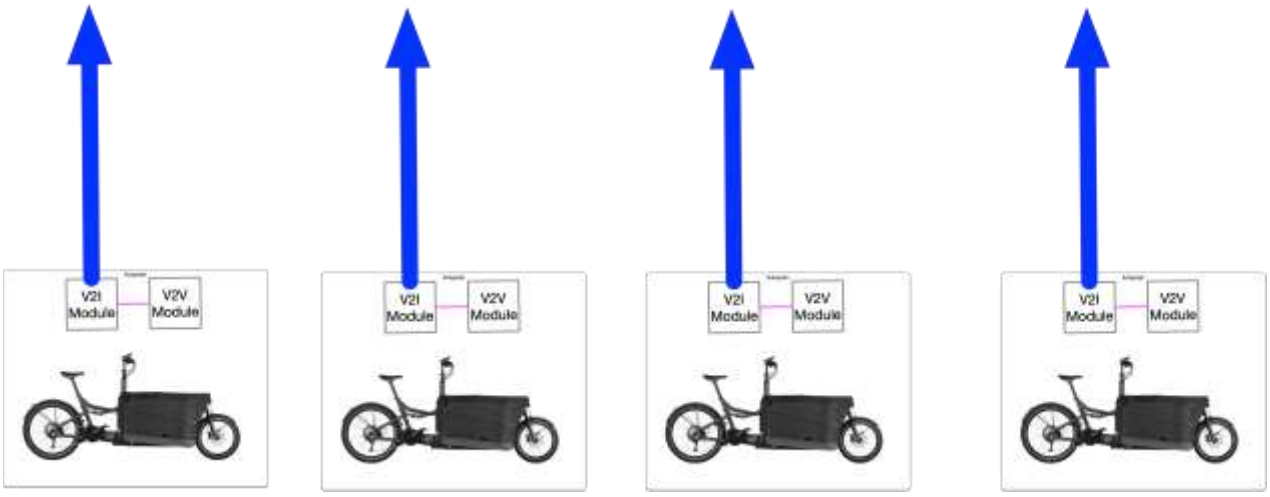

Figure 25: V2V firmware upgrade using V2I communication

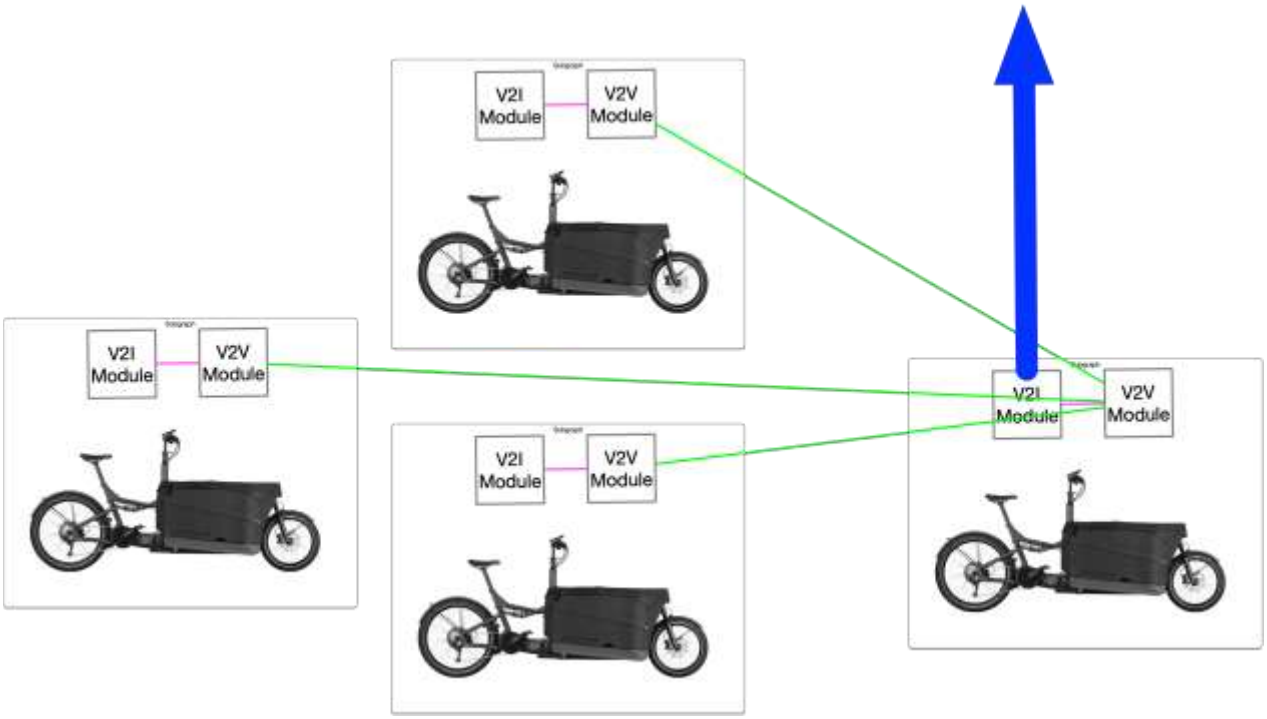

Figure 26: V2V firmware upgrade using V2V communication

3.2.3 V2V communication Security

Since the FW update images are digitally signed (using a private ECDSA key, and verified upon boot (using the corresponding public key) before activation, no additional measures for image integrity during the transmission will be taken. The bootloader, which un-modifiable, will act as a root-of-trust. The secure bootloader was built using TF-M software, an open-source resource for the STM32U5 and other TrustZone-enabled STM devices. Specifically, it utilizes MCUboot for the secure boot part.

The signing of the images will be handled by a specialized AWS service called “AWS Signer”. It ensures that the private key used cannot be retrieved by any means after its upload. The generation of the key however remains the responsibility of the developer. Each time a new FW version is prepared, it is put in an AWS bucket, which triggers a lambda function that will perform the MCUboot-compatible signature and the launch of the FUOTA. The aforementioned cloud infrastructure can be deployed using the cloud development kit along with our scripts.

The update process typically involves the cloud provider, in our case AWS, to launch an IoT Job on specific Things. During the thing's creation, an authentication key may be generated. A thing can be hence seen as a remote system connected to the AWS cloud using a unique AWS IoT registered thing's key. The V2I module is expected to contain such an authentication key to securely connect to the AWS IoT Core. Therefore, each Thing will be listed in AWS, allowing a precise tracking of each bike’s FW version. The IoT Jobs are used to trigger any remote action, compatible with the system, to a set of Things (individual Things or an IoT group). The Things are responsible for listening for new Jobs on a specific MQTT topic. Additionally, things are required to provide Job progress reports to the cloud. The MQTT topics, and update progress API, are managed by the V2V module, as part of its security mission.

The V2I module has been mocked by a Raspberry Pi to allow for the development. The Raspberry PI, connected to the internet, communicates with the V2V module over UART using the UART protocol defined in Section 3.2.4.

The integration with the actual V2I module needs to be carried out. The spreading of the FUOTA to the sibling nodes over TDMA (FUOTT for short), still needs to be implemented. Yet, in no scenario is a middleware device, held for example by the manufacturer, needed. To handle the FUOTT, a fleet of bike will register to the master bike (a bike with internet access). Once a FUOTT is available, the master bike performs the update for himself first. Once the update is deemed reliable, the sibling bikes (not

connected to internet, but to the master bike via FUOTT) will report their presence to the master bike. After some delay, the master bike report, to AWS, the available bikes that are about to perform the update. Then, the update is sent in a BROADCASTING mode. Packets, which where lost during transmission, are re-sent by the master. Once the transmission to all responsive bikes is done, the master bike triggers a reboot of the fleet and collects the status of each device. The master bike is also responsible to relaying the FUOTT progress to AWS.

3.2.4 V2V UART protocols

The STM32U5 needs to have access to a MQTT proxy over UART (the V2V-V2I link). Each instruction, sent to and from the STM32U5, is encoded using CBOR (https://github.com/intel/tinycbor). Each CBOR message is prepended with a separator (0x8c,0x4f,0x2d,0x73,0x42) and a length field (two bytes, big-endian).

The CBOR messages themselves contain the MQTT commands, formatted according to the following table (taken from https://docs.aws.amazon.com/freertos/latest/userguide/freertos-ble-library.html#freertos-ble-gatt-profile).

Message Type	Message	Map with these key / value pairs
0x03	PUBLISH	• Key = "w", value = Type 0 Integer, Message type (3) • Key = "u", value = Type 3, Text String, Topic for publish • Key = "n", value = Type 0, Integer, QoS for publish • Key = "i", value = Type 0, Integer, Message Identifier, Only for QoS 1 Publishes • Key = "k", Value = Type 2, Byte String, Payload for publish
0x04	PUBACK	• Sent Only for QoS 1 messages. • Key = "w", value = Type 0 Integer, Message type (4) • Key = "i", value = Type 0, Integer, Message Identifier
0x08	SUBSCRIBE	• Key = "w", value = Type 0 Integer, Message type (8) • Key = "v", value = Type 4, Array of text strings, topics for subscription • Key = "o", value = Type 4, Array of Integers, QoS for subscription • Key = "i", value = Type 0, Integer, Message Identifier
0x09	SUBACK	• Key = "w", value = Type 0 Integer, Message type (9) • Key = "i", value = Type 0, Integer, Message Identifier • Key = "s", value = Type 0, Integer, Status code for Subscription
0X0A	UNSUBSCRIBE	• Key = "w", value = Type 0 Integer, Message type (10) • Key = "v", value = Type 4, Array of text strings, topics for

Message Type	Message	Map with these key / value pairs
		unsubscription • Key = "i", value = Type 0, Integer, Message Identifier
0x0B	UNSUBACK	• Key = "w", value = Type 0 Integer, Message type (11) • Key = "i", value = Type 0, Integer, Message Identifier • Key = "s", value = Type 0, Integer, Status code for UnSubscription

3.2.5 V2V TDMA protocols

The data exchanged between vehicles via Vehicle-to-Vehicle (V2V) communication will be transmitted over the 2.4GHz radio band using a Time-Division Multiple Access (TDMA) protocol. This protocol ensures that all vehicles can send and receive data without overlapping transmissions by assigning specific time slots for each vehicle to communicate.

The radio network consists of two types of nodes:

1. Multiple Sensor Nodes (SNs): Generally considered as data generators.
2. A single Wireless Data Concentrator (WDC): Collects data from various SNs and transmits it via the Vehicle-to-Infrastructure (V2I) link.

The network is organized in a star topology, where each vehicle contains a Sensor Node connected to a single Wireless Data Concentrator. The WDC receives data from all SNs and transmits it to the V2I link, while also sending commands (e.g., configuration or orders) to the SNs.

This architecture centralizes complexity in the WDC, allowing SNs to use low-power, low-footprint solutions, typically powered by batteries, whereas the WDC may require a proper power supply.

The TDMA-based V2V link can be used to transmit data from vehicles to the infrastructure (e.g., collected data) or from the infrastructure to the vehicles (e.g., pairing bikes, sending instructions, or steering commands).

The “cell” (comprising the WDC and all connected SNs) shares a single maximum throughput, as they use the same radio channel. The radio has a 2Mbps transmission rate, but due to necessary downtimes and collision avoidance, the estimated current uplink transmission rate is roughly 250Kbps. The downlink

throughput is limited, and the TDMA protocol assigns slots in 100ms periods, leading to potential latencies of over 100ms if a message is corrupted.

Potential improvements include:

- Increasing throughput: With changes, the usable throughput could reach 1Mbps, supporting both uplink and downlink data, potentially meeting the requirements of a FUOTT.
- Optimizing latency: Reducing the TDMA period to smaller values (e.g., 10ms) or optimizing slot allocation could achieve more reasonable latencies, suitable for steering vehicles via radio.

3.3 TRACE, MASA and V2I connectivity

To enable delivery status updates and general status communication with the TRACE infrastructure (StreamHandler), MASA's Modena server infrastructure and V2I connectivity will be leveraged. The V2I module will employ an MQTT broker to facilitate communication between vehicles (such as cargo bikes and drones) and MASA, utilizing 4G/5G networks for real-time data exchange on vehicle status and delivery progress. The key data points shared through MQTT messages include:

- Vehicle status, GPS coordinates (latitude, longitude, height)
- Timestamp, camera and object IDs
- Speed, orientation, and navigation type
- Residual kilometers, loaded and maximum weight, operator presence
- Special transport status
- Destination coordinates for both ground and flying vehicles.

The V2I connectivity with MASA will facilitate the availability of current vehicle status data for the TRACE platform, enabling efficient tracking of shipments and vehicle operations.

An overview of the communication environment between the entities, that integrates with V2I communication module, MASA and TRACE is represented down below:

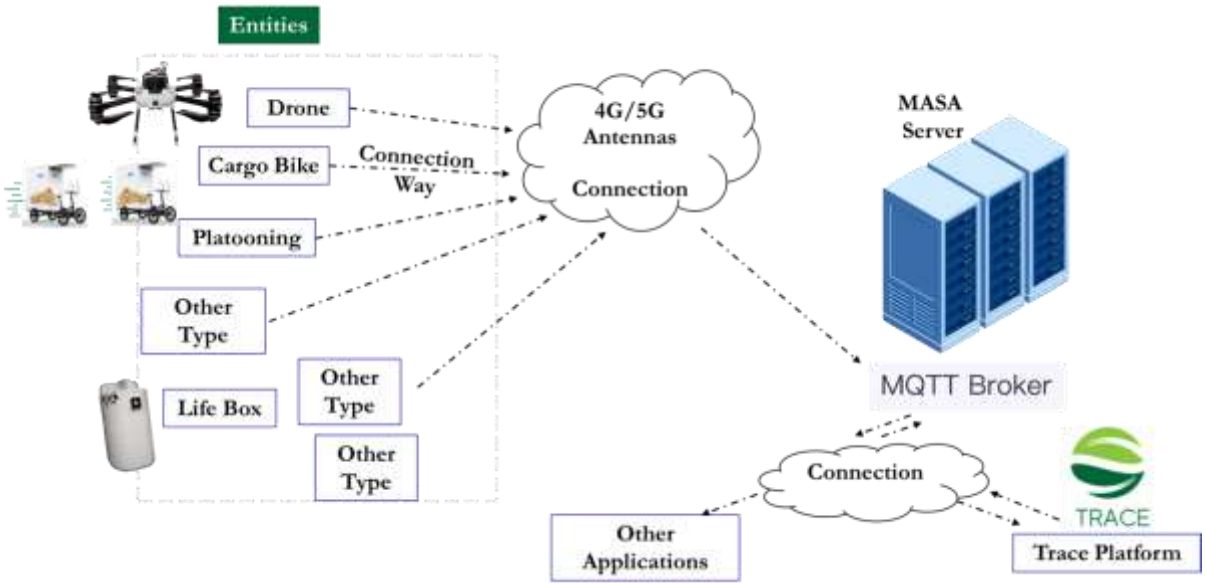

Figure 27: Connectivity diagram between Entities, Masa, Trace and other application

To optimize vehicle status updates for the TRACE platform, MASA implements periodic status reports at predefined intervals, avoiding the need for continuous operator tracking if an operator is present (e.g., updates every 3 seconds, adjustable as needed). MASA might also streamline data by sending only essential information, omitting potential redundant details like destination coordinates when TRACE has pre-defined the route. Additionally, MASA could enable optional real-time transmission of Cargo Bike camera frames for other applications, contingent on available bandwidth.

3.4 RB-VOGUI Interface

In this section, we will explain the process of establishing a connection between the TRACE architecture and the RB-VOGUI robot. This integration will be facilitated through the Kafka communication protocol, which serves as a middleware for data exchange. The protocol will extract data from the robot, including sensor readings and state information, and publish it through various topics. This structure allows the TRACE architecture to efficiently subscribe to relevant data streams for real-time analysis and decision-making.

Additionally, the Kafka protocol will be utilized to send commands back to the robot, enabling it to execute a range of actions based on the architecture's directives. This bidirectional communication is

crucial for ensuring the robot operates effectively and responds to changing conditions, ultimately achieving the desired behavior in its tasks.

To enable effective communication with the robot, the following Kafka topics will be used:

- rbvogui_robot_state: This topic provides information about the robot's current status, including its operational state. It allows for real-time monitoring of the robot's condition, that could be:
    - Robot_status:
        - standby: the robot is ready to receive orders. Power saving behaviors can be applied
        - ready: the robot is ready to work or working
        - emergency: the robot cannot work/operate correctly temporarily. It can imply a recovery procedure or an external action from the operator
        - failure: the robot is not working. It needs the operator intervention and probably restart/reboot the robot
    - Operational_status:
        - idle: the robot is doing nothing
        - calibrating: the robot is calibrating
        - moving: the robot is moving
        - raising: the elevator is raising
        - lowering: the elevator is lowering
        - charging: the robot is charging the battery

For example, when the robot will be moving the received output will be:

Robot State: standby, Operation State: moving

- rbvogui_robot_odom: This topic provides detailed information about the robot's movement and speed through odometry data. It tracks the robot's position, orientation, and linear and angular velocity over time, allowing the system to monitor its real-time motion. The received topic will contain:
    - Robot_odom_pose
        - Position: x, y, z. Measures the translation from the initial point.

        - Orientation: x, y, z, w. Measures the rotation from the initial point
    - Robot_odom_velocity
        - Linear: x, y, z. Measures the current linear velocity.
        - Angular: x, y, z. Measures the current angular velocity.

  For example, when the robot will have moved one meter in X-axis, with a velocity of 0.5 meter per second in that direction, the received output will be:

```
Robot_odom_pose: Position: x: 1.0, y: 0.0, z: 0.000000, Orientation: x: 0.000000, y: 0.000000, z: -0.284775, w: 0.958595, Robot_odom_velocity: linear: x: 0.500000, y: 0.000000, z: 0.000000, Angular: x: 0.000000, y: 0.000000, z: 0.000000
```

- rbvogui_robot_battery_data: This topic provides the current battery level of the robot, allowing real-time monitoring of its power status. For instance, when the robot will have a 90% of battery, the received output will be:

```
Level: 90.0000
```

- rbvogui_robot_safety: This topic indicates whether the robot is in an emergency stop due to safety reasons. The received topic will contain:
    - Emergency Stop: returns true if the emergency stop button has been pressed.
    - Safety Stop: returns true when the robot has stopped due to a safety reason.
    - Detecting Obstacles returns true if some obstacle has been detected or false if not.

  For instance, if the robot detects an obstacle and stops because of security reason, the received output will be:

```
Emergency Stop: false; Safety Stop: true; Detecting Obstacles: true
```

- rbvogui_robot_gps_data: This topic provides the GPS values of the robot in terms of latitude, longitude, and the covariance of the current GPS coordinates. The received topic will contain.
    - Latitude: GPS latitude coordinates.
    - Longitude. GPS longitude coordinates.
    - Covariance: GPS covariance that represents the uncertainty in position.

For instance, a possible received output could be:

```
Latitude: 39.551597, Longitude: -0.468524, Covariance: [0.000225, 0.0, 0.0, 0.0, 0.000225, 0.0, 0.0, 0.0, 0.00040]
```

- rbvogui_robot_rgbd_image: This topic allows for the extraction of raw data from the compressed image captured by the robot and converted to binary data to send all messages in almost real time.

To send action commands to the robot, the following Kafka topics will be used:

- rbvogui_robot_teleoperation: This command facilitates teleoperated control of the robot, allowing velocity values to be published to achieve forward, backward, and rotational movements. Specifically, linear velocity is applied along the x-axis, and angular velocity is applied around the z-axis, while maintaining the remaining parameters at zero. The published topic message should be:

```
{"linear": {"x": <linear_mov>, "y": 0.0, "z": 0.0}, "angular": {"x": 0.0, "y": 0.0, "z": <angular_mov>}}
```

  For example, to send a movement command of 1 meter per second while rotating at 0.5 radians per second, the message would have the following form:

```
{"linear": {"x": 1.0, "y": 0.0, "z": 0.0}, "angular": {"x": 0.0, "y": 0.0, "z":0.5}}
```

- rbvogui_robot_remote_stop: Through this communication topic, it is possible to send a command that halts the robot, acting as a remote stop button. To send this command, a message must be published through this topic with a value of True to stop the robot, or False to allow it to continue functioning. For instance, if there is necessary to completely stop the robot, the published message with this topic should be:

```
True
```

3.5 StreamHandler Data Management

The data management will be run as a dockerized API service. This component will allow data to be stored and accessed to the TRACE storage for any downstream tasks. The component will serve as a robust interface facilitating the receipt and transfer of data between the interoperability layer and the StreamHandler module. This interface ensures seamless communication and data exchange, enhancing overall system efficiency. The StreamHandler leverages two distinct data storage mechanisms depending on the nature and format of the data being processed. Specifically, TRACE selects between object storage and document storage based on the data type to ensure optimal storage and retrieval performance.

To identify and categorise data accurately, the component employs the Multipurpose Internet Mail Extensions (MIME) identifier[6]. This common framework allows for the precise detection and classification of various data types, streamlining the storage process. For object storage, the system utilises MinIO, a highly scalable and efficient solution for storing unstructured data such as multimedia files or large data sets. On the other hand, MongoDB is used for document storage, providing a flexible NoSQL database ideal for structured and semi-structured data storage.

The data management component is implemented as a Python wrapper, which can be accessed via a user-friendly web interface. This design choice ensures flexibility and easy integration with existing web-based applications, offering a seamless user experience. System users will be able to access the platform using traditional password-based authentication methods or a more advanced decentralized wallet as a blockchain-based authentication mechanism. This integration ensures a secure and seamless login experience without requiring the storage of private keys on the platform. This dual-layered approach provides enhanced security, ensuring that data remains protected from unauthorised access.

Data storage processes are designed with efficiency and retrieval in mind. The system will extract, and store relevant metadata associated with each data entry. This metadata serves as an index, making it easier for users to locate and retrieve information when needed. By employing a structured approach to data storage and retrieval, the component maximises both accessibility and security, enhancing the overall user experience.

[6] *Johnson, K (2000). Internet Email Protocols: A Developer's Guide. Addison-Wesley Professional. ISBN 978-0-201-43288-6.*

3.6 Storage and management of local data (on-board)

The most essential component of the TRACE platform is the on-board data storage and management system, which enables the proper collection, organization and local storage of data from all sensors and environmental detection units deployed on vehicles and telemetry systems. This functionality aims at continuous operation in the mobile logistics network for supporting real-time monitoring and response objectives of TRACE and could be deployed either on robots or on the appropriate processing units that could be placed on the robots.

On-board data management starts with the real-time collection and organization of various types of data, such as status indicators of vehicles, sensor records from environmental conditions, route adherence data, and alerts that triggered by faults or other anomalies. Data is organized and temporarily held in on-board storage modules, optimized for large data volumes typical in high-frequency logistics environments. By pre-organizing data, the on-board system will ensure that everything is stored and ready for fast transmission to the TRACE platform as soon as network connectivity becomes available.

In contrast to traditional logistics systems that depend only on periodic uploads of data, TRACE's on-board storage system is designed to maintain the continuous read data, supporting real-time transmissions without mediators in vehicle operations. This becomes even more important in low-connectivity areas where data might have to be stored locally until transmission becomes possible. Hence, a well-structured repository of recent vehicle and environmental data is maintained within the system, supporting seamless data delivery to the TRACE back end when the connectivity is reestablished to ensure that no operational insights are lost and event detection with timely scheduling by TRACE is performed.

This scaled management of the data enhances the efficiency of the system in that vital information is brought in time under close examination, while storage space is optimally used for continued operation. TRACE will be able to capture and store key logistics data in all types of conditions with this powerful on-board data management framework, ensuring a constant flow of information to increase the level of real-time decision-making and operational efficiency across the platform.

3.7 Network infrastructure provisioning

Considering the network infrastructure layer, the identified technology blocks at the stakeholders' side include the end-user devices (sensors along with their controlling equipment), edge compute nodes hosting any near-to-end-user functions, and edge/mesh network equipment; while at operator network side the communication infrastructure blocks include the core network functions and the underlying compute infrastructure along with any storage infrastructure. The exact network elements of this layer depend highly on the selected communication technologies. On their turn, the selected network deployment options for the TRACE demos, depend highly on the public network availability at the selected sites and at the capability of the partners to access these networks, as well as on the capability of the partners to deploy non-public networks. To this end, for the Italian and Slovenian demonstration sites existing 4G/5G communication networks will be used, while for the Greek telco sites, the existing 4G/5G Non-Stand-Alone network of OTE will be used for the Chalkis and Thessaloniki demo sites (Greek Use Cases/ storylines 1 and 2), while for the third Greek demonstrator alternative network deployments and sites are investigated.

3.7.1 Communication Infrastructure Layer for Italian Demonstrator

The network infrastructure layer that will support the Italian Use Case, will be based on 4G/5G network deployment. The infrastructure that will be used for the communication system, involving different transportation entities, including drones, cargo bikes, and platooning systems is illustrated in Figure 27.

In the context of this Use Case connectivity between the end-devices and vehicles and the MASA Server will be established over existing 4G/5G network infrastructure. The MASA server processes data, relaying it through an MQTT Broker that interface with the Trace Platform for further processing and applications. Additionally, roadside equipment or smart devices, represented as "Other Type", can feed data into this system, ensuring integration between vehicles and the platform for real-time logistics tracking.

3.7.2 Communication Infrastructure Layer for Slovenian Demonstrator

The network infrastructure layer that will support the Italian Slovenian Use Case, will be based on existing 4G/5G (outdoor and indoor) and Wi-Fi (indoor with limited 4G/5G connectivity) network architecture/ deployment, to replicate real-world conditions in city centers and real use cases of mass usage of autonomous delivery robots.

Existing 4G/5G and Wi-Fi networks will ensure low-latency and high-reliability of data transfer for seamless communication and synchronization. This will include support for autonomous navigation capabilities, dynamic routing, and coordination of vehicles, based on real-time environmental and operational data. By adhering to privacy and data protection, the network infrastructure usage demonstration will showcase a scalable solution.

The infrastructure that will be used for the communication system, involving different transportation entities, such as autonomous delivery robots (ADR), cargo bikes, and electric delivery vehicles is illustrated in the following figure.

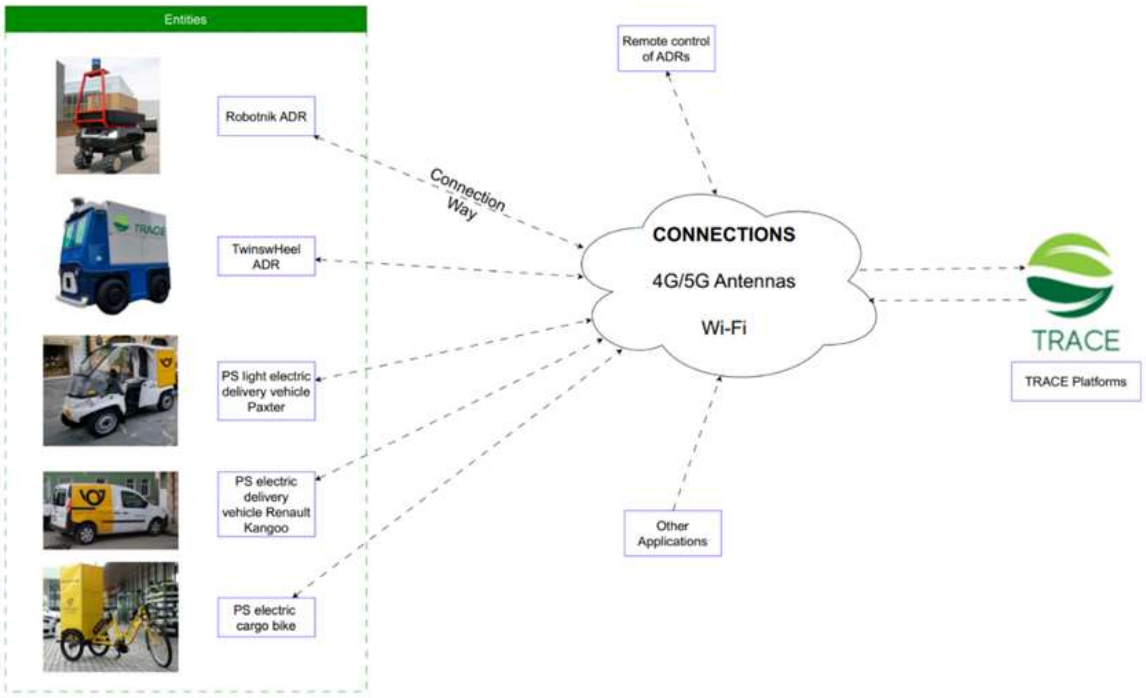

Figure 28: Communication Infrastructure Layer for Slovenian Demonstrator

3.7.3 Communication Infrastructure Layer for Greek Demonstrator

The network infrastructure layer that will support the Greek Use Cases, will be based on public 4G/5G network deployment, while Wi-Fi connectivity can be also an option depending on the devices network interfaces. The infrastructure that will be used for the communication system, involving different transportation entities, including cargo trains, trucks and last mile vehicles (drones, autonomous vehicles), roller cages, etc. is illustrated in the following figure.

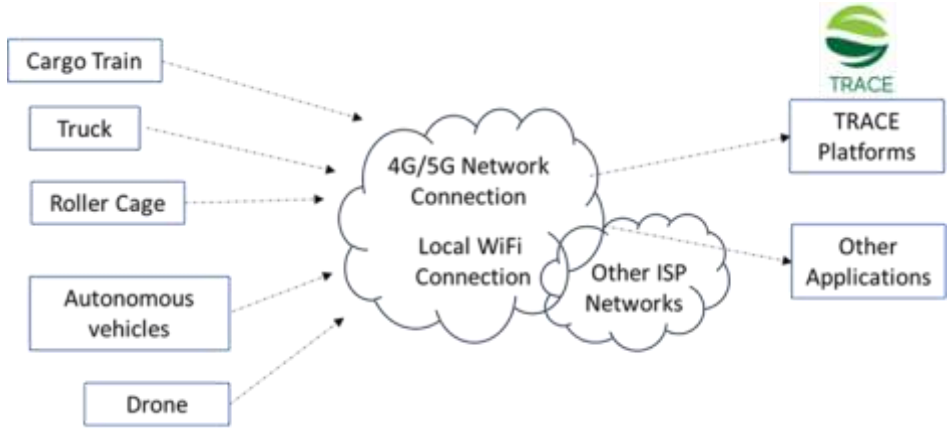

Figure 29: Communication Infrastructure Layer for Greek Demonstrator

In particular, the vehicles and end devices of this Use Case communicate via OTE public 4G/5G network (network implementation is based on ERICSSON mobile network equipment) that is deployed at the ACS and selected HT station sites in Thessaloniki, Chalkis and Athens areas. This connectivity enables communication from the end devices to the NKUA Cloud Infrastructure where the TRACE Platform will reside for data processing and application provisioning. Connectivity between the devices or/and the TRACE Platform with external / other applications will be performed over the 4G/5G network and the interconnected ISP networks (DNs) – OTE's ISP network.

As far as the third UC/storyline of the Greek demonstrator is concerned, the network infrastructure to be deployed will depend on the current availability of the OTE public network coverage, and the granted permits for deploying network infrastructure in campus – where the administrative activities are currently in process. The identified campus sites are the NKUA and the OTE Academy premises as an alternative location. On availability of public network coverage the relevant network infrastructure will be based on ERICSSON public mobile network equipment. In the case of a separate network deployment

in the form of NPN (non-public-network) the OTE 5G (SA) Stand-Alone testbed network will be utilized (With the Core Network elements hosted in LeonR&Do lab at OTE-Academy premises).

The LeonR&Do lab comprises both 4G/5G NSA and 5G SA network deployments, that are integrated with carrier-grade RAN equipment:

- The hybrid 5G NSA testbed (Figure 30) is composed of a lightweight 4G/5G EPC/IMS core network (2 VMs on a Dell R630 server) from ATHONET, MEC implementation via SGW-LBO (Local BreakOut), two NOKIA Airscale 4G/5G BTSs, eight NOKIA 4G/Wi-Fi Flexi-Zone Multiband Indoor Pico BTS, supporting standard network interfaces (such as S1 and X2), 5/10/15/20 MHz LTE carriers with 2x2 MIMO along with edge/core cloud infrastructure for deploying/hosting a number of edge services/applications.
- The 5G SA testbed (Figure 31) composed of 5GC/IMS from ATHONET (deployed on Redhat Openstack Cloud Infrastructure), three BB6630 NR BBU, six ERICSSON 4408 RUs (4x4 MIMO, 100MHz, 3.5GHz (N78)), six ERICSSON Dot(s) 4479 (4x4 MIMO, 100MHz) and 2 UPFs to emulate edge and core 5G network Data planes, incl. 3GPP Control Plane Network Functions, exposing N1, N2, N3, N4 and N6 interfaces.

Both testbeds are being connected to the internet via a 10Gb/s broadband connection (over GRNET). DL data rates of ~1150 Mbps, UL Data rates ~85 Mbps and E2E max. latency of ~ 5ms (towards internal iperf and external servers) have been demonstrated using 5G SA UEs and CPEs. The 5G SA deployment can be configured to support various flavors of network slicing.

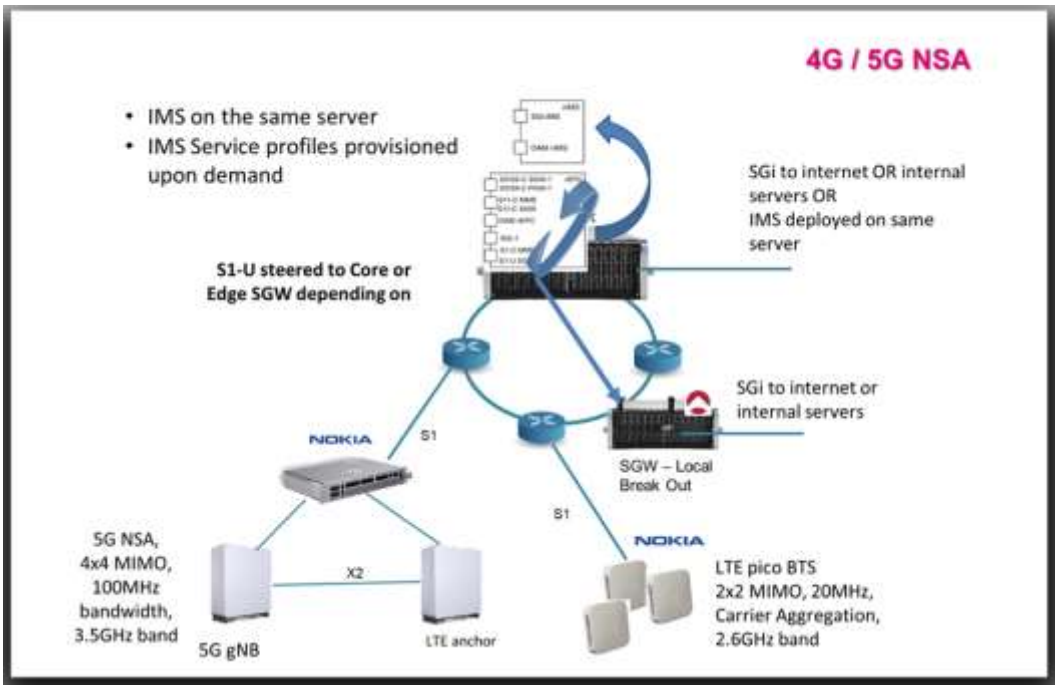

Figure 30: 4G/5G NSA testbed deployment

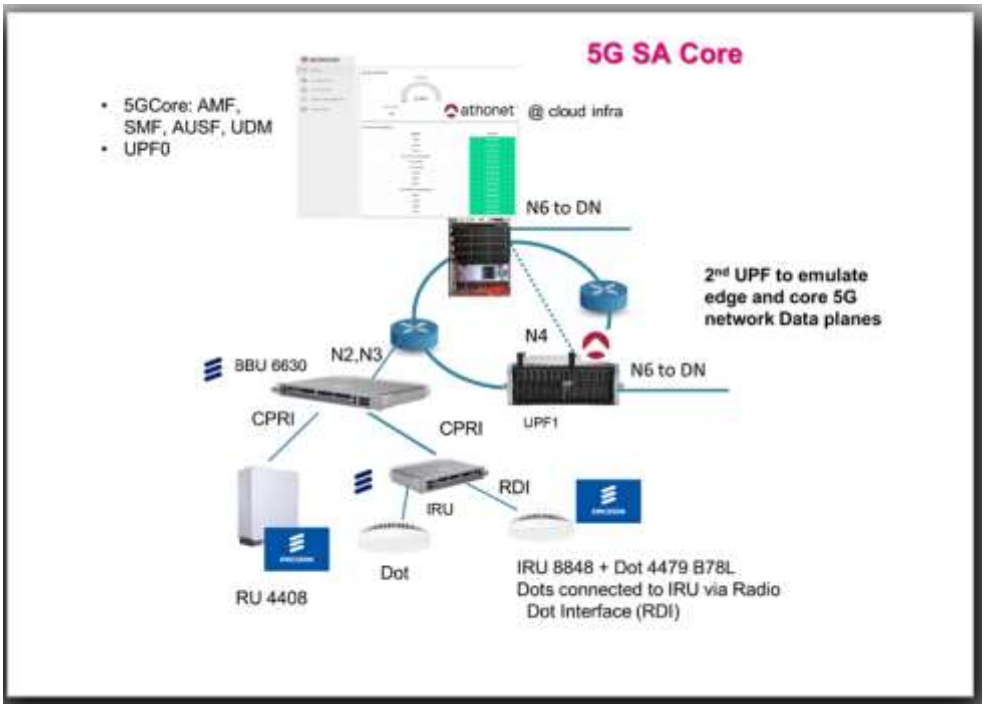

Figure 31: 5G SA testbed deployment

Depending on the granted permits for radio access network infrastructure deployment ERICSSON gNBs will be deployed in the defined campus site to support the relevant demo activities.

3.8 Connection with the several infrastructure data sources used in the pilots

The TRACE pilots aspire to demonstrate the potential of innovative logistics solutions using state-of-the-art infrastructure data sources, sensors, and real-time communication systems. Pilot activities take advantage of various data inputs to optimize logistics operations, resource allocation, and delivery efficiency at all instances while allowing transparency with blockchain-enabled records. The pilots focusing on micro-hub operations and urgent parcel deliveries via drones implement multi-modal last-mile delivery with cargo bikes and drones, integrating synchromodal operations and autonomous delivery solutions. At the heart of every pilot is a fully integrated link between vehicles and sensors through the TRACE platform, using the Streamhandler system, which enables real-time data synchronization, route optimization, and improved coordination of operations.

In these logistics' operations, many data sources are used. The data that comes from these data sources are monitored continuously through their integration into Streamhandler and constitute the base for effective decision-making, the seamless operation of the components of the TRACE platform that use these data and, especially they are used for the optimization of logistics processes. In the pilots there are the following types of data sources:

- Company Information Data: These data give information about each company has been registered in TRACE including a Company ID as it was created by TRACE, the name, the location and the contact information for the company.
- Spatial Data: These data include information about the areas that each company which is registered in TRACE platform operates. These data are the ID of an area as it is created by TRACE, the type of the area and which logistic company is responsible for an area. Also, data like the latitude, longitude and altitude are included for specific points in an area.
- Vehicle Data: These data come from sensors that are placed on the vehicles and from TRACE platform. More specifically, the data include the ID and the type (e.g. truck, train, etc.) of the

vehicle according to the pilot, number of bikes in a platoon (Italian pilot only), the vehicle status (in delivery or ready to be loaded for new delivery), the current location as it is indicated by GPS coordinates, the current and the maximum speed of the vehicle, the fuel consumption per 100Km (both actual and projected) and the average distance that a vehicle can travel when is fully loaded. Additionally, there are sensors that project the fuel/battery status and the operational status of the vehicle as well as operational cost for a vehicle as well as the operational time for loading and unloading of a vehicle. Also, in the data are included the load capacity in two aspects, from the aspect of volume (length, width and height) and from the aspect of weight of the vehicle.

- Shipment data: In these data are included information like the type of shipment (letter, envelope, box, roller cage etc.), the priority of the shipment, its status (e.g., sensitive, normal etc.), its ID as it defined by TRACE, its volume, weight and actual size. Also, the shipment data gives information about the Origin and Destination Location using GPS coordinates, the date and time that a shipment picked up by the responsible logistic company as well as the date and time the end user received it. In addition, there are data that characterize the estimated Arrival Time/Date (from origin hub to destination hub), the constraints about shipments, the Vehicle ID in which a shipment is loaded and lastly the logistic company which is responsible for the shipment. Moreover, the name of the sender and the recipient will be included too, as well as the shipment RFID, the delivery preferences and the customer satisfaction feedback.

- Environmental data: A set of sensors provide these data giving a view about the environment around and inside the vehicle. More specifically, the sensors monitor the conditions at which the shipments are delivered and transported (e.g. temperature, humidity etc), the pedestrian density, the traffic conditions and the air quality index. In addition, camera sensors are used to give visual information about the status of shipments during the transportation, detection of obstacles, cyclists etc.

- Routes Data: These data give information about the ID of the vehicle, the mode (truck, railway, air, etc.), the start and the delivery date and the estimated arrival time. Also, route data can provide information about the path through GPS checkpoints and the shipments that follow a route.

- Event Detection data: This kind of data comes from the TRACE platform. Especially, these data include the ID of the event and its type, the location that event took place, and the affected shipments based on their status.
- Smart Contract data: These data are produced by the smart contracts giving information about the IDs of origin company, carrier, consolidation agent, middle mile logistics agent, first mile pick-up agent and last mile delivery agent. Also, information for the Owner company and Carrier company, the price and the payment type are provided. Furthermore, data for delivery vehicle information, the services performed by the agent and the delivery performance report are included.

3.9 Integrate Real-Time Information and Intelligent Re-Routing Mechanisms

3.9.1 Traffic Management System (TMS)

In contrast to traditional traffic-unaware routing, which relies primarily on static road network information and fixed speed assumptions, traffic-aware routing incorporates real-time or historical traffic conditions to produce more reliable travel time estimations. This distinction is particularly important in logistics operations, where delivery schedules, vehicle utilization, and KPI achievements are directly influenced by route accuracy. By integrating traffic intelligence, routing engines can proactively adjust planned itineraries to more efficient operations.

Traffic management systems that expose real-time or historical measurements for a delivery area or an entire city can provide significant value to last-mile operations, where the traffic situation during the delivery window is unlikely to change drastically. In these cases, using current traffic conditions improves the estimation of arrival times. Conversely, historical traffic data will be utilized to generate informed predictions for locations farther away or for middle-mile planning, where real-time data are not available at the time of scheduling. Such historical data enables the system to form a roughly estimated educated guess about expected road speeds and congestion conditions.

The platform is designed to be able to ingest traffic-related data originating from sensors or known measurement points along the road network. These may include geospatial coordinates of road

segments, timestamps of observations, and average vehicle speeds recorded at each location. By incorporating this information into the routing computation, the system will be capable of selecting the most efficient route for delivery vehicles, particularly when the requested schedule concerns immediate pickup or delivery scenarios. In these cases, the predictive capability and accuracy of the traffic data become essential factors in determining optimal routing.

As a proof of concept, traffic data provided by Region of Attica will be utilized to demonstrate and validate the integration of traffic-aware routing within the Athens metropolitan area. This initial implementation will serve as a reference scenario for including traffic APIs, assessing system performance, and the overall impact on routing efficiency. In future phases, TRACE should identify and cooperate with additional data providers capable of supplying comparable traffic information, enabling the deployment of this functionality across other cities and operational environments.

The integration of traffic-aware capabilities will be implemented in the Route Optimizer during the second half of the project's lifetime. The approach, methodology, and corresponding architectural extensions will be documented in the upcoming project deliverables.

3.9.2 Weather Management System (WMS)

Weather conditions significantly influence the operational capabilities, safety, and efficiency of vehicles used in logistics operations. To ensure robust and adaptive mission planning, TRACE has developed a weather-aware vehicle suitability framework that evaluates vehicle readiness under specific environmental conditions (proof of concept developed for the first version of the system, while the fully functional module will be developed and deployed as a service for the second and final version of the system). For the implementation of this system a custom RSWT rating scale for the weather suitability of each vehicle has been implemented, tailored to the needs of the project. The system extracts real-time or forecasted meteorological data, depending on the reference time of the operation's deployment, and triggers a filtering mechanism that determines which of the available vehicles can safely operate in the given operational conditions. The modular architecture of the system supports the use of different weather sources. The final version of the module, that will be deployed in the TRACE system, has been designed and will be built with the use of the EO4EU services, enabling alignment with European open-data and interoperability standards. The outcome is a capability-aware fleet selection mechanism that enhances operational resilience, regulatory alignment, and overall operational efficiency. This

mechanism will be integrated into the system's Fleet Manager module via REST API calls, enabling it to automatically filter the available vehicles for each logistics operation before forwarding the eligible set to the optimization algorithms (Scheduler modules).

To characterize vehicle tolerance to environmental conditions, we have defined the RSWT scale, which quantifies allowable ranges for:

- R: Rain (mm/h)
- S: Snow (cm/h)
- W: Wind (km/h)
- T: Temperature (°C)

Each component is assigned a value 1–5, where the lower the number of the rating (i.e., 1) the vehicle is considered to be very robust, with a wide operational range, while the larger the number of the rating (i.e., 5) the vehicle is considered to be very limited, having a narrow operational range.

Table 4: RSWT Scale Ranges

Scale	Rain (mm/h)	Snow (cm/h)	Wind (km/h)	Temperature (°C)
1	0–50	0–5	0–80	-40 to +60
2	0–30	0–3	0–60	-20 to +50
3	0–15	0–2	0–40	-10 to +40
4	0–7	0–1	0–25	0 to +30
5	0–2	0	0–10	+5 to +20

In the TRACE system, each vehicle is registered in the database along with a RSWT rating value that prescribes its operational capabilities, according to standardized qualification or empirical operational knowledge. According to Table 1, a vehicle with a RSWT rating of 3124 will be able to operate under moderate rain (7–15 mm/h), light snow (4–5 mm/h), medium wind (25–40 km/h), and moderate temperatures (0–30 °C).

This weather-aware vehicle suitability framework will be deployed as a REST API service, that will receive as input the list of vehicles, along with their RSWT ratings, a reference location, that should correspond to either the depot position of the logistics operation, or an indicative location in the wider operation's

region, and optionally, a reference time, in case that the operation is scheduled for a different time than the one that the call was performed.

Vehicle Registration

As mentioned above, each vehicle registered in the TRACE database should have a RSWT rating, based on either the manufacturer's specifications or operational experience.

Weather Data Retrieval

The weather-aware vehicle suitability framework will retrieve the meteorological conditions for the given location and time, by calling a weather API. This step will allow the easy integration of different services for weather data, with the only limitation of providing the required information in any form.

Weather–Vehicle Matching and Filtering

The service will compare the weather conditions corresponding to the rating of the vehicles, to the conditions retrieved by the weather API. All the vehicles that exceed any weather threshold will be excluded from the list and will not be able to participate in the logistics operation.

Mission Planning

The list of the appropriate vehicles will be forwarded to the corresponding services, so that they can be used for the optimization procedure and participate to the logistics operation.

The integration of the weather-aware vehicle suitability framework within the TRACE system will increase the operational safety, reliability and efficiency of the operations, offering automatic adaptation to changing weather conditions, reducing the manual decision workload and the possibility of human errors with significant potential negative impact on the logistics operations.

A minimal demonstrator has been built using the OpenWeatherMap API to validate the RSWT logic. This implementation includes (i) weather data retrieval, (ii) a filtering mechanism according to weather conditions, (iii) a threshold checking mechanism for the filtering of the fleet, and (iv) an example of fleet filtering for a given location. A Python implementation of this prototype can be found in ANNEX A. The system has been tested for various locations, weather conditions, and vehicles with different ratings. An indicative output, as printed by the system, for a point with the following geographic coordinates can be found below:

Given input:

- Latitude: 63.5340
- Longitude: 8.6545
- List of vehicles:
  - truck_01 – RSWT: 2131
  - van_02 – RSWT: 3124
  - scooter_01 – RSWT: 5545

Produced output:

- Rain (1h): 0.0 mm
- Snow (1h): 0.3 cm
- Wind Speed: 2.9 km/h
- Temperature: -0.1 °C
- Suitable vehicles: ['truck_01']

In the operational system, weather data retrieval will be replaced by an EO4EU-based interface, enabling:

- Querying climate data cubes such as ERA5-Land, CAMS, and ECMWF forecasts
- Extracting variables for specific coordinates and timestamps
- Computing aggregates (e.g., hourly max rain, daily mean wind)
- Using harmonized formats (NetCDF, JSON, GeoTIFF)

Expected Inputs

- Latitude and longitude
- Reference timestamp or interval
- Required variables: precipitation, temperature, wind, snow depth
- Dataset ID (e.g., ERA5_LAND_HOURLY)

Expected Outputs in JSON structure

- "rain": 0.2,
- "snow": 0.0,

- "wind": 3.8,
- "temp": 7.3

This integrates seamlessly into the RSWT filter.

The modular weather-aware vehicle suitability framework described above allows the integration of standardized weather data for the optimization of logistics operations. By employing a custom RSWT rating for the weather suitability of the vehicles, and utilizing weather data deriving by various sources, using a modular design, the system integrates the weather suitability of the vehicles into the overall operational optimization procedure, leading to safer, more efficient and resilient operations. The planned transition to EO4EU as the primary weather-data provider further enhances system interoperability, aligns the service with EU-standardized Earth Observation data infrastructures, and creates a foundation for reuse or extension by other EU projects and initiatives pursuing similar objectives

3.10 Standards and Data Interoperability Framework

The TRACE platform has been built with international standards and principles that allow seamless data exchange between their components and systems that are outside the platform.

3.10.1 Sensor Layer

Sensors are linked to the platform by means of Raspberry Pi modules that act as edge nodes and are charged with data collection and subsequent transmission to the platform. These Raspberry Pi modules receive data such as temperatures, location data, current speed data ,air quality and pressure and forward them to the platform in JSON format by means of HTTP/HTTPS RESTful APIs. These activities are carried out in line with the ISO/TC 154 (Data Interchange) and ISO 8000 (Data Quality) standards.

3.10.2 Communication Standards

The data exchange between sensors, edge devices, and the platform occurs through 4G/5G networks that adhere to 3GPP and ETSI standards. The above mentioned standards ensure that data transmission over 4G/5G networks has high speeds with lower latency. Furthermore end-to-end encryption with TLS 1.2/1.3 ensures data privacy is maintained during data exchange with the cloud and other modules.

Additionally the data exchange adheres to ISO/IEC 27001 (Information Security) and ISO/IEC 27701 (Privacy Information Management) standards.

3.10.3 API - Data Exchange and Messaging Framework

The use of OpenAPI specifications enables defining and specifying RESTful APIs to be integrated seamlessly with other components. JSON format is utilized to exchange data to ensure compatibility and uniform data processing. Additionally, the platform is integrated with Apache Kafka to enable asynchronous messaging and operations that are standard-compliant with ISO JTC 1 and ETSI. The use of the Flask web framework in the implementation of the APIs makes it possible to create flexible communication services. The Swagger interface provides the standardized layer that ensures that there is adherence to the principles of interoperability.

3.10.4 Data ingestion through Excel

To improve user experience and facilitate the process of inputting data in large numbers to the platform data in bulk from Excel documents (XLS/XLSX) can be imported directly from the Graphical User Interface (GUI). Users are also allowed to upload data through GUI, (for example: vehicles, sensors and locations), eliminating the need to manually input data. Validation processes are carried out during imports to check the data format and quality in line with ISO 8000 (Data Quality) and ISO/TC 154 (Data Exchange) with the data format converted to JSON.

3.10.5 Data Privacy

Within the TRACE solution, the blockchain integration service operates according to international norms to guarantee security, traceability, and interoperability. The blockchain layer relies on the Algorand blockchain network, which supports the ARC-19 standard and manages the creation and handling of the dynamic NFTs (dNFTs), as mentioned above. The standard supports the creation of mutable metadata, which is guaranteed by the smart contract provided by the blockchain technology developed by Algorand. The handling of data privacy relies on an encryption algorithm named AES-256 before loading the data onto IPFS (InterPlanetary File System), and each uploaded file receives a Content Identifier (CID).

3.10.6 Weather Data Integration

The TRACE platform incorporates weather information in a way that supports day-to-day operational decisions rather than simply storing raw data. To achieve this, it uses a dedicated Weather Suitability Service that collects real-time or forecasted conditions from external providers. In the early prototype, the team relied on the OpenWeatherMap API because it allowed quick testing of key parameters such as rainfall, snow, wind, and temperature. As the system matures, this setup will be replaced with an integration based on the EO4EU framework. Through EO4EU, TRACE gains access to Copernicus and ECMWF datasets using a standardized interface that follows European open-data and interoperability practices. Once retrieved, the weather information is converted into JSON and compared automatically with the RSWT ratings stored for each vehicle. This process makes it possible to judge instantly whether a vehicle can safely operate under the expected conditions, adding an extra layer of reliability to operational planning.

3.10.7 Traffic Data Integration

Traffic information is handled in a similar spirit. Instead of treating it as an isolated data feed, TRACE brings it into the platform through traffic management APIs that offer either current measurements or historical performance of specific road segments. These inputs may include congestion levels, typical speeds or time-based observations that describe how traffic behaves at different hours of the day. After being collected, the data are reshaped into a uniform JSON structure so that the Route Optimizer can use them directly. Real-time feeds help refine arrival-time estimates and improve routing accuracy when operations are scheduled immediately, while historical data become especially useful when live updates are not available. For the first demonstration, the platform will incorporate traffic information provided by the Region of Attica, allowing the team to validate the traffic-aware routing logic in an actual metropolitan setting.

4 Interfacing and Integration with other TRACE Modules

4.1 Alignment with the TRACE Data Model

Building upon the infrastructure, vehicles, and sensor technologies discussed in the preceding chapters, this section illustrates how those elements map onto the TRACE Data Model. Each component, ranging from cargo bikes and drones to on-board sensors and communication modules, corresponds to one or more classes within the model. By adhering to standardized attributes, data types, and relationships, the platform ensures consistency, interoperability, and ease of data exchange across diverse logistical scenarios. Below, we examine the main classes most relevant to the discussed infrastructure and explain how their attributes are populated with real-world data from the vehicles, sensors, and operational processes highlighted earlier.

4.1.1 Vehicles and the Vehicle Class

All vehicles introduced in the previous chapters (cargo bikes, RB-VOGUI ground robots, trains, trucks, and UAVs) are represented in the model by the *Vehicle* class. Key attributes include:

- VehicleID (int): A unique identifier assigned by TRACE. For example, the cargo bike used in the Italian demonstration might be *VehicleID=101*, while a drone could be *VehicleID=102*.
- VehicleType (string): Distinguishes between Train, UAV, UGV, Van, Truck, Bike, etc. This aligns with the variety of transportation modes in TRACE.
- VehicleStatus (string): Tracks a vehicle's current operational state. Examples from the demonstrations include "Operating" (actively delivering), "Onhold" (waiting at a hub), "Damaged" (in need of repair), or "Idle."
- Shareable (boolean): Indicates whether the vehicle can be shared across multiple logistics operators or is exclusively dedicated to a single company at a time.
- OperationalCost (float): Expressed in Euro/hour. This ties into each vehicle's cost model—particularly useful for real-time optimization or cost estimation in scheduling.
- NominalConsumption (float): Fuel or energy consumption rate (e.g., kWh/hour for electric cargo bikes or liters/hour for trucks).

- NominalRange (float): Estimated maximum distance the vehicle can travel on a full charge or full tank.
- NominalSpeed (float): Average operating speed, used for route planning.
- FuelType (string): Identifies “Gasoline,” “Electricity,” or other energy sources.
- LoadServiceTime / UnloadServiceTime (float): Time in seconds required for loading/unloading.
- VehicleExternalID (string): Allows each logistics provider to reference its own internal ID system.

During system initialization, data about each specific vehicle—collected from procurement records or existing fleet management systems—is entered into the TRACE platform, populating the *Vehicle* table. Updates to *VehicleStatus* or *NominalRange* occur dynamically as real-time sensor data or operational events (e.g., refueling, recharging) become available.

4.1.2 Real-Time Monitoring via RealTimeVehicleInfo

Dynamic attributes such as a vehicle’s instantaneous location, current fuel level, velocity, and immediate fuel consumption appear in the *RealTimeVehicleInfo* class:

- VehicleInfoID (int): A unique identifier for each record.
- VehicleLocation (string): WGS84 coordinates ([lat, lon]) received from on-board GPS modules.
- FuelLevel (float): The current battery percentage (for electric vehicles) or remaining fuel percentage (for combustion vehicles).
- Velocity (float): The vehicle’s current speed in meters/second, as measured by sensors (e.g., wheel encoders, GPS speed).
- FuelConsumption (float): Instant consumption, e.g., kWh/hour or liters/hour.

Vehicles continuously update this class through the communication infrastructure described in Chapter 3. When a cargo bike moves, for example, the on-board controller pushes the updated *Velocity* and *FuelLevel* to the TRACE platform via MQTT or other protocols. This data is then stored as a new *RealTimeVehicleInfo* record, allowing dispatchers and optimization modules to monitor operational states in near real-time.

4.1.3 Infrastructure, Areas, and Waypoints

The preceding chapters also described pilot sites and routes (e.g., micro-hubs, railway stations, charging points). These are represented in the model primarily by the *Area* and *Waypoint* classes:

- Area:
    - *AreaID (int)*: Unique identifier for each area, such as "1" for a central logistics hub.
    - *Responsible (string)*: Name of the logistics company or facility manager overseeing that region.
    - *AreaType (string)*: Identifies if an area imposes "Restrictions" (e.g., low-emission zones) or "Permissions" (authorized loading/unloading).
- Waypoint:
    - *WaypointID (int)*: Unique identifier for each routing checkpoint.
    - *WaypointLocation (string)*: The lat/lon coordinates in WGS84 format.
    - *Characteristics (string)*: A textual descriptor like "Train Station," "Gas Station," or "Charge Point."
    - *Coordinator (string)*: The responsible entity for that waypoint (a hub operator, city authority, or logistics partner).

When planning or executing a *Journey*, a vehicle may pass through multiple *Waypoints*. The platform populates these classes with data from pilot site surveys and geolocation providers. The *Responsible* or *Coordinator* attributes, in turn, map to an organization listed in the *Company* class, ensuring consistent referencing and permissions management.

4.1.4 Shipments and Goods

The concept of shipments introduced in the earlier chapters—such as parcels transported by cargo bikes or heavier goods moved by rail—map to the *Shipment* class. Relevant attributes include:

- ShipmentID (int): Created by TRACE to uniquely identify each package or consignment.
- Origin / Destination (string): Postal or structured addresses for pickup and drop-off.

- ShipmentWeight / ShipmentVolume (float): Overall mass (kg) and volume (cm^3), crucial for load planning.
- ShipmentType (string): Standard or fragile shipments, influencing handling requirements.
- ShipmentStatus (string): “Ongoing,” “Rejected,” or “Processing,” among other states.
- PickUpDate / DeliveryDate (datetime): Actual timestamps for collection and final handover.
- Priority (int): Allows ranking of urgent shipments over routine ones.
- RespLogisticCo (string): The responsible carrier or partner.
- ScheduledDateDelivary (datetime) and DeliveryTimeWindow (int): Indicate the upper limits for timely delivery, supporting dynamic scheduling and real-time route recalculations.
- ShipmentExternalID (string): The shipping company’s internal reference.

If a *Shipment* consists of multiple items (e.g., individual parcels or palettes), those items appear in the *Good* class. *Good* captures additional details such as item weight, volume, and any special handling requirements (e.g., refrigeration). This split model allows for finer granularity when shipments contain different categories of items, each possibly requiring unique conditions.

4.1.5 Load Management: The Load Class

For vehicles like cargo bikes or trucks, the *Load* class helps to track capacity constraints for every vehicle:

- LoadID (int): Created by TRACE to uniquely identify the load unit.
- WeightCapacity / VolumeCapacity (float): The maximum mass or volume the load container can accommodate.
- Dimensions (string): Physical dimensions of the load container (length, width, height).
- LoadSpecialReq (string): Extra features such as temperature control or hazardous-material compliance.
- Stable (boolean): Whether the load container can secure items against movement.
- LoadType (string): The type of container or unit (Refrigerator, Trailer, Roller Cage, etc.).

- FullyLoaded (boolean): Indicates no additional space is available.

In practice, a cargo bike might have *LoadID=201* (a small cargo box with defined capacity), whereas a truck might have *LoadID=202* (a much larger trailer). As *Shipments* get assigned to a vehicle, the system checks if the *ShipmentWeight* and *ShipmentVolume* fit within the *WeightCapacity* and *VolumeCapacity* of the target *Load*. If capacity is exceeded, the logistics planner either arranges for a second vehicle or readjusts assignments in real time, using the platform's resource management modules.

4.1.6 Events and Operational Disruptions

When a sensor or operator detects a malfunction, delay, or damage, the platform registers it as an *Event*:

- EventID (int): Created by TRACE.
- EventType (string): E.g., "Malfunction," "Damage," or "Delay."
- Affect (string): Specifies whether the disruption pertains to a *Shipment*, a *Vehicle*, or a *Load*.

For instance, if a cargo bike's servo motor fails, an *Event* record is created with *Affect=Vehicle* and *EventType=Malfunction*. The platform can then invoke contingency measures, such as dispatching a replacement vehicle or reassigning the shipment to another resource.

4.1.7 Capturing Routes with Journey and TransportationMode

A *Journey* encapsulates the operational span during which a vehicle is assigned to one or more shipments:

- JourneyID (int): Unique to each journey instance.
- StartDate / EndDate (datetime): Capture the interval from departure to arrival.
- JourneyStatus (string): Often "Ongoing" or "Idle," but can also store states like "Completed" or "Aborted."

Vehicles are further linked to *TransportationMode* records—e.g., "Air," "Road," "Rail," or "Water"—which specify overarching characteristics like permissible speed or regulatory constraints. By defining these relationships, TRACE can better optimize route planning, especially when combining different vehicle types (drone deliveries with ground cargo bike staging, for example).

4.1.8 Populating the Model with Data

Data flows into the model through both manual configuration and automated processes:

1. Initial Setup: Administrators and system integrators define base records and static data that do not change frequently, e.g., *Company* entries, *Area* definitions, *Waypoint* coordinates, and known *Vehicle* attributes—using import scripts or user interfaces.
2. Sensor Data Ingestion: As explained in Chapter 3, each vehicle's on-board systems periodically send updates on velocity, location, and consumption to the TRACE platform, populating or updating *RealTimeVehicleInfo* entries.
3. Logistics Processes: Shipment records are created via the TRACE GUI or modified at various stages (e.g., pick-up, sorting, and delivery), reflecting real operational events.
4. Event Generation: Automatic triggers or manual operator alerts create *Event* entries whenever anomalies occur, ensuring the system can replan and adapt quickly.

4.2 Interaction between sensing modules, communication infrastructure, and data processing components

The core aspect of the TRACE platform involves interaction among sensing modules, communication infrastructure, and data processing components, whereby the logistic data from vehicles and environmental sensors continues to flow to the central TRACE system for monitoring and analysis. The system collects real-time data coming from different sensing modules and allows TRACE to adjust its logistics operations on time. The on-board vehicle and infrastructure sensing module collect vehicle status, environmental conditions, and positional tracking data critical to real-time logistics oversight. The data is transmitted directly to TRACE, ensuring that no local processing might take place and consequently all information will be easily available centrally for event detection and scheduling. In this respect, by routing data quickly to the central TRACE platform, the system supports real-time situational awareness and decision-making throughout the logistics network.

StreamHandler enables the communication between sensing modules and TRACE by using Kafka for efficient, high-throughput data streaming and RESTful APIs to enable flexible, scalable data integration. StreamHandler is an intermediary that organizes the incoming data and manages the flow of information

in such a way that high-priority data, such as vehicle status updates or environmental alerts, are available immediately for analysis. This centralized approach ensures TRACE can handle large volumes of data efficiently, ensuring a continuous flow of data and therefore supporting efficient logistics operations. Consolidating all data onto the TRACE platform enables the system to work in real-time event detection for logistics, dynamically adjusting schedules and routes according to changes. The centralized processing components of TRACE evaluate the incoming streams of data for patterns, deviations, and other operational triggers, enabling TRACE to make data-driven decisions about optimizing the flow of logistics work. This interaction framework combines direct data capture by sensing modules, efficient data streaming via StreamHandler, and API-based data integration into one holistic and resilient system, empowering TRACE to maintain operational functionality and adapt changes in logistics for better efficiency of the overall network. Through these interconnected layers, TRACE is able to monitor and manage logistics operations in a diverse and dynamic environment, ensuring the platform is robust and responsive.

5 Conclusion

Throughout this deliverable, the multifaceted challenge of creating a robust, scalable sensing infrastructure and resource management framework for TRACE was addressed. Ensuring uninterrupted data flow among diverse vehicles (cargo bikes, UGVs, UAVs, trains, trucks), sensors, and communication protocols demanded innovative methods for hardware integration and low-latency networking. Solutions involved selecting appropriate sensor modalities, designing power-efficient systems, and employing dynamic offloading strategies for computation-heavy tasks. These measures ensure both continuity of service in unpredictable environments and cost-effectiveness in day-to-day logistical operations.

A key outcome of this work is a comprehensive blueprint covering various aspects from vehicle capabilities and on-board data handling to end-to-end communication pathways. We have demonstrated how the vehicle-to-vehicle and vehicle-to-infrastructure interfaces can work in tandem to share information about load status, location, fuel or battery levels, and more—ultimately enabling data-driven decisions and real-time route optimization. Additionally, we explored modular resource management approaches that adapt to shifting operational conditions, striking a balance between local processing on vehicles and offloading to edge/cloud nodes.

By integrating these infrastructural elements and communication channels into the broader TRACE platform architecture, we pave the way for advanced features like autonomous platooning, real-time visibility of cargo conditions, and event-triggered route re-planning. This deliverable thus constitutes a critical building block for the project, informing subsequent modules responsible for data analytics, interoperability, and higher-level decision-making in multi-modal logistics scenarios.

Moving forward, upcoming steps include further validation of these infrastructures in real testing environments (i.e., the three TRACE pilot sites), final refinement of communication protocols and security safeguards, and enhancements to data processing workflows to accommodate larger fleets.

ANNEX A: Python Implementation of Integration Weather Management System

```
import requests

api_key = "insert_your_key_here"

lat = 61.404553969478904
lon = 7.873537213826781

# ------------------------
# 1. Vehicle definitions
# ------------------------
vehicles = [
    {"id": "truck_01", "RSWT": "2131"},
    {"id": "van_02", "RSWT": "3124"},
    {"id": "scooter_01", "RSWT": "5545"},
]

# RSWT thresholds
RSWT_MAP = {
    "rain":   [50, 30, 15, 7, 2],    # mm/h
    "snow":   [5, 3, 2, 1, 0],       # cm/h
    "wind":   [80, 60, 40, 25, 10],  # km/h
    "temp":   [(-40,60), (-20,50), (-10,40), (0,30), (5,20)]  # °C
}

# ------------------------
# 2. Weather fetching function
# ------------------------
def fetch_weather(api_key, lat, lon):
    """Fetch weather from OpenWeatherMap (example)."""
    url = f"http://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={api_key}&units=metric"
    response = requests.get(url).json()

    print(response)

    weather = {
        "rain": response.get("rain", {}).get("1h", 0),
        "snow": response.get("snow", {}).get("1h", 0),
        "wind": response.get("wind", {}).get("speed", 0),
        "temp": response.get("main", {}).get("temp", 18)
    }

    print(f"\n  Current Weather Conditions for location ({lat:.4f}, {lon:.4f}):\n")
```

```
    print(f"  Rain (1h):    {weather['rain']:.1f} mm")
    print(f"  Snow (1h):    {weather['snow']:.1f} cm")
    print(f"  Wind Speed:   {weather['wind']:.1f} km/h")
    print(f"  Temperature:  {weather['temp']:.1f} °C")
    print("-" * 45)

    return weather

# -------------------------
# 3. Vehicle filtering function
# -------------------------
def is_vehicle_suitable(vehicle, weather):
    RSWT = vehicle["RSWT"]

    # Rain
    rain_limit = RSWT_MAP["rain"][int(RSWT[0])-1]
    if weather["rain"] > rain_limit:
        return False

    # Snow
    snow_limit = RSWT_MAP["snow"][int(RSWT[1])-1]
    if weather["snow"] > snow_limit:
        return False

    # Wind
    wind_limit = RSWT_MAP["wind"][int(RSWT[2])-1]
    if weather["wind"] > wind_limit:
        return False

    # Temperature
    temp_min, temp_max = RSWT_MAP["temp"][int(RSWT[3])-1]
    if not (temp_min <= weather["temp"] <= temp_max):
        return False

    return True

# -------------------------
# 4. Filter fleet
# -------------------------
def filter_fleet(vehicles, weather):
    suitable = [v for v in vehicles if is_vehicle_suitable(v, weather)]
    return suitable

# -------------------------
# 5. Example usage
# -------------------------
```

```
if __name__ == "__main__":
    # Example: Use dummy weather or call API
    weather = fetch_weather(api_key, lat, lon)

    fleet = filter_fleet(vehicles, weather)
    print("Suitable vehicles:", [v["id"] for v in fleet])
```