

# SMART and FLEXible mobile DATA COLLECTOR for GIS

(Acronym, MOBILO)

ENTERPRISES 0916/0055

[D14] Report on the development of the  
 software that is used during the video  
 collection process

<b>Deliverable n.</b>	D14	<b>Deliverable title</b>	Report on the development of the software that is used during the video collection process
<b>Workpackage</b>	WP6	<b>WP title</b>	Route Planning – data gathering GUI
<b>Editors</b>	Elias Frentzos (GEO), Dimitrios Skarlatos (CUT)		
<b>Contributors</b>	Elias Frentzos (GEO), Dimitrios Skarlatos (CUT), Petros Katsikadakis (GEO), Kyriakos Toumbas (GEO)		
<b>Status</b>	Final		
<b>Distribution</b>	Public		
<b>Issue date</b>	2020-07-31	<b>Creation date</b>	2020-07-01

## Contents

LIST OF FIGURES .....	3
LIST OF TABLES .....	4
LIST OF ABBREVIATIONS.....	5
REVISION CHART AND HISTORY LOG.....	6
1 Introduction.....	8
2 General Software Development architecture .....	8
2.1 Specialized APIs that were developed for the MobiloGrabber application.....	9
2.2 Existing Components .....	9
2.3 Newly developed Components .....	10
2.4 General design of MobiloGrabber Application and hardware components.....	10
3 GeonoesisGPS library .....	11
3.1 Library description.....	12
3.2 Namespace NMEA.....	12
3.3 Namespace NTRIP .....	14
3.4 Namespace GeonoesisGPS.....	17
4 VideoGrabber library.....	19
4.1 Library description.....	19
4.2 VideoGrabber Classes.....	19
5 Map Matching and Routing.....	21
5.1 Map Matching Algorithm .....	21
5.2 Routing algorithm performance.....	22
6 MobiloGrabber Application.....	23
6.1 General Description.....	24
6.2 Produced files .....	26
7 Conclusions.....	27
8 References .....	29

## LIST OF FIGURES

Figure 1: Data flow in MobiloGrabber.....	10
Figure 2: Class (UML) Diagram of GeonoesisGPS library.....	11
Figure 3: Class (UML) Diagram of VideoGrabber library .....	18
Figure 4: Synthetic road network. Different edge colours indicate different group of edges	22
Figure 5: Part of a CPP example solution (segments 91 to 213) .....	22
Figure 6: Chinese Postman algorithm execution time vs number of edges in the graph .....	23
Figure 7: Example screen of MobiloGrabber software during data collection .....	24
Figure 8: Example screen of MobiloGrabber software during system’s initialization .....	25
Figure 9: Example data contained in a MOBILO project folder.....	27

## LIST OF TABLES

Table 1: Overview of Ublox’s F9P NMEA sentences processed by GeonosisGPS .....	12
Table 2: Annotation used for classes’ properties and methods.....	13
Table 3: Properties and Methods of GPGGA Class .....	13
Table 4: Properties and Methods of GPGLL Class .....	13
Table 5: Properties and Methods of GPGGA Class .....	14
Table 6: Properties and Methods of GPGST Class .....	14
Table 7: Properties and Methods of GPGSV Class.....	14
Table 8: Properties and Methods of NTRIPNetwork Class .....	15
Table 9: Properties and Methods of NTRIPCaster Class.....	15
Table 10: Properties and Methods of NTRIPDataStream Class.....	16
Table 11: Properties and Methods of NTRIPclient Class .....	16
Table 12: Properties and Methods of SerialPort Class .....	17
Table 13: Properties and Methods of GPShandler Class.....	18
Table 14: Properties and Methods of VideoGrabberBase Class .....	20
Table 15: Properties and Methods of VideoGrabberHardware Class .....	20
Table 16: Properties and Methods of VideoGrabberThread Class.....	21
Table 17: Project files containing actual on-site data. ....	27

## LIST OF ABBREVIATIONS

<b>API</b>	Application Programming Interface
<b>DLL</b>	Dynamic Link Library
<b>SDK</b>	Software Development Kit
<b>GPS/GNSS</b>	Global Positioning System / Global Navigation Satellite System
<b>INS/IMU</b>	inertial navigation system / inertial measurement unit
<b>RTK</b>	Real Time Kinematic
<b>PPS</b>	Pulse Per Second
<b>CPP</b>	Chinese Postman Problem

## REVISION CHART AND HISTORY LOG

REV	DATE	REASON
0.1	01/07/2020	Initial
0.2	15/07/2020	Draft
0.3	31/07/2020	Final

## Executive Summary

This deliverable describes part of the results of MOBILO's project WP6. This WP primarily aims at developing a software component that is used by the system's end user before and during the data collection process. Specifically, as suggested by the project proposal, the component should at least provide the following functionality (a) Display a map, (b) Gather Video Data (c) Gather GPS / INS Data. This software component is very essential to the end user to assist her during the – demanding in terms of focalization - video collection process: record and mark the already visited road network parts, while at the same time supervising the video collection process and adjusting possible interface parameters. The software developed by this WP can be used to pair with the MOBILO hardware components, such as GPS RTK data, INS and Cameras. Specifically, the data gathering developed software, utilizes all components connected to a laptop with windows 10 and a wireless connection to the internet. The software assures the synchronization between all connected modules, that is, the GPS-RTK2 board, MTi-7 INS/GNSS and FLIR (machine vision) cameras.

## 1 Introduction

This WP primarily aims at developing a software component that is used by the system's end user during the data collection process. Specifically, as suggested by the project proposal, the component provides the following functionality:

- Display a map
- Gather Video Data
- Gather GPS / INS Data

To achieve this functionality the software component should be able to connect to all MOBILo system's parts, handle streamed data and store them in the hard disk of the underlying computer in a synchronized manner. Therefore, the developed software component should handle both positional and rotational data by the positioning subsystem, as well as image data streamed by the imaging subsystem.

Moreover, given the current position of the sensors, the software should be capable to display the system's current position on a map, as well as all the recorded so far trajectory of the respective vehicle. The software should also demonstrate the *current* status of the imaging subsystem, by displaying several frames of the gathered video data so as to give the user the ability to check whether all components are satisfactory working. Summarizing, the following indications are displayed over the UI of our software:

- GPS / GNSS Condition (fix / float / autonomous / timeout)
- MTi-7 connection status (connected / timeout)
- Machine vision Cameras (gathered frames are displayed, e.g., 5 sec)

Regarding the map used in the background, our software can display vector data (polygons, lines etc.), raster maps, as well online XYZ tile – based maps, such as google maps, Bing maps, OpenStreetMap etc. As such, the user has the ability to choose among several backgrounds which are suitable for her needs. The software also supports several projection systems for the geographically displayed data, including WGS84, UTM, Cyprus's LTM etc. On the other hand, all data collected are stored in the application's data folder in the underlying system on which data are collected, i.e., WGS84.

Although the final output of this WP is an application with a friendly user interface capable of supporting the data collection process, we have developed two underlying libraries (APIs) that support this interface. This option was dictated by the need for code reuse, as well as the possible universal usages of several components developed during the project's execution. For example, *GeonoesisGPS* library developed under this work package, is used in our *MobiloGrabber* application, while it can be used for any GPS / GNSS related application that exploits u-blox F9P board [1]; in fact, we have already developed a proof-of-concept application that uses F9P board and *GeonoesisGPS* library to perform simple topographic measurements, a by-product of our project.

## 2 General Software Development architecture

The primary output of this WP is *MobiloGrabber* application that is used during the video collection process. While this is desktop application with a user interface, it is based on several C#, VB and C++ projects that provide several components used throughout all



developed applications. The components are provided in the form of dynamic link libraries which can be categorized into three categories:

- APIs developed during MOBILO project's evolution regarding the project's core, such as libraries responsible for communication with GPS / GNSS devices, etc.,
- APIs provided by external vendors and / or open-source components that are used in our APIs (of the first category) and / or applications.
- APIs developed during MOBILO project's evolution that can be reused in a variety of other applications outside of MOBILO project.

In the next sections we further analyze the components used throughout the application development life cycle.

## 2.1 Specialized APIs that were developed for the MobiloGrabber application.

Regarding the API developed during MOBILO project that is used by MobiloGrabber application, these constitutes of the following projects that form dynamic link libraries (dll) files:

- **GeonoesisGPS C# project:** GeonoesisGPS provides a library with objects capable of communicating with GPS / GNSS devices via a virtual com port (usb device), configure them according to the user needs, receive and parse NMEA sentences to produce meaningful information, send RTCM corrections to the device provided by a NTRIP caster and record raw data transmitted by the receiver. The respective project provides a class library in the form of .NET dynamic link library.
- **VideoGrabber C# project:** VideoGrabber provides a library with objects capable of communicating both synchronously and asynchronously with FLIR cameras, configure them, start exposing and grabbing pictures on a timely manner, whereas trigger can be provided by both software and hardware sources.
- **MobiloBase C# project:** This is the main output of WP7; it provides all functionality used to manage georeferenced video streams constructed by the MobiloGrabber application. Provides geometry objects and algorithms, spatiotemporal objects (trajectory), photogrammetry objects and procedures, as well as handling of video data synchronized with the object's trajectories. Its usage in the current context is limited to the usage of spatiotemporal objects in a unified manner with all other applications.

## 2.2 Existing Components

There are also several external APIs that are used inside the developed applications. Among them, the most important are:

- **NettopologySuite [2]:** A .NET GIS solution that is fast and reliable for the .NET platform. NetTopologySuite is a direct port of all the functionalities offered by JTS Topology Suite: NTS expose JTS in a '.NET way', as example using Properties,



Indexers etc. The JTS Topology Suite is a Java library for creating and manipulating vector geometry. It also provides a comprehensive set of geometry test cases.

- **SharpMap** [3]: An easy-to-use mapping library for use in web and desktop applications that provides access to several vector and raster data sources.
- **SpinnakerNET** [4]: SpinnakerNET is part of the FLIR API used to communicate with the cameras of the imaging subsystem, configure them and handle the produced images. SpinnakerNET is used in .NET applications.
- **SpinVideoNET** [4]: SpinVideoNET is part of the FLIR API used to create avi (video) files with .NET applications.

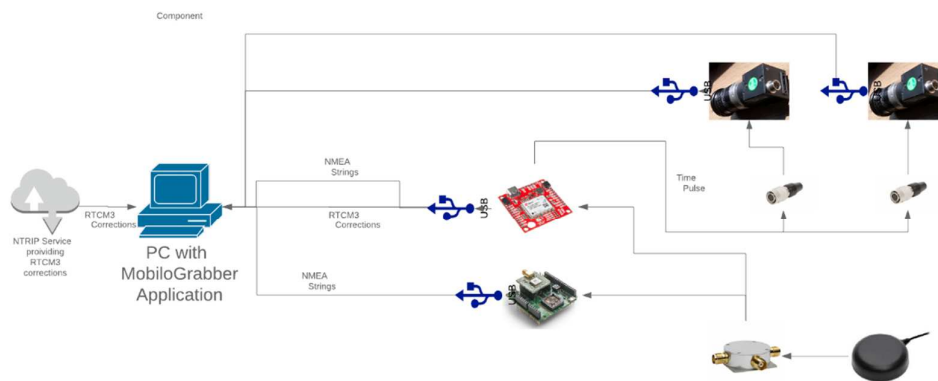
### 2.3 Newly developed Components

Inside our project we have also included APIs from several external libraries developed by us during the project's execution which are isolated from the rest of the project, since they provide general functionality that can be reused in several applications. These features are common among several software projects that are developed within the project's organizations:

- **GeonoesisWinForms**: API that provide general functionality for winforms such as zoomable picture boxes, supporting of drawing objects in picture boxes, forms' settings retrieval and update, form stylization etc.
- **GeonoesisMaps**: API that provides general mapping utilities, such as coordinate system transformations, topology and cleanup operations, grid management, providers for xyz tiles consumed on all maps displayed on the developed applications, reading, and writing shapefiles etc. [D13]

### 2.4 General design of MobiloGrabber Application and hardware components

Implementing the developments of WP3 and WP4 in our context, we have designed the data flow that is displayed in Figure 1 and is supported by the MobiloGrabber application.



**Figure 1:** Data flow in MobiloGrabber

Specifically, MobiloGrabber application connects to the positioning and imaging subsystems of MOBILO system via USB interfaces. Both GPS / GNSS and INS / IMU boards connect to an L1 / L2 GPS / GNSS antenna through a signal splitter. GPS / GNSS connects through GPIO cables to the cameras so as to pass the hardware trigger signal provided by its TP / PPS interface. Here we must note that a less accurate software trigger can be also used to trigger the cameras, which is usually used in lab conditions, e.g., to take pictures of a chessboard for calibration purposes. Finally, MobiloGrabber application synchronizes all hardware components, connects to an NTRIP caster via the internet to send RTCM corrections to the GPS / GNSS, and receives and stores in the disk all data provided by them.

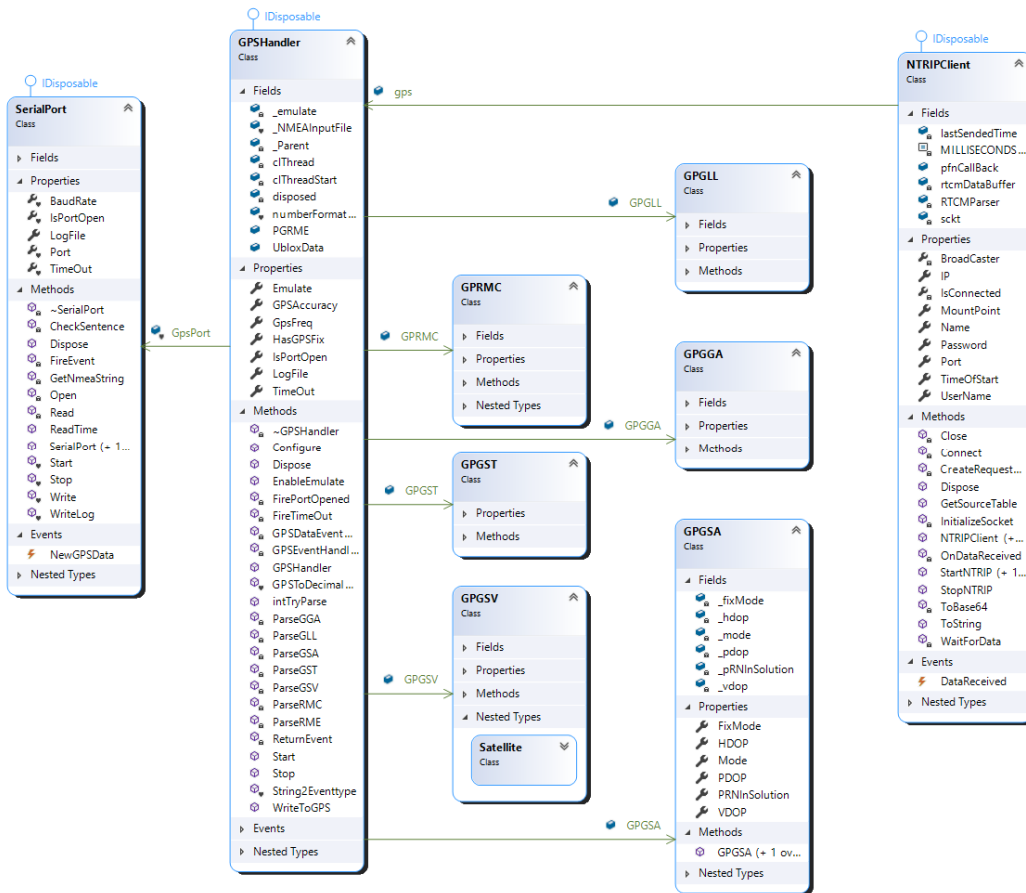


Figure 2: Class (UML) Diagram of GeonosisGPS library

### 3 GeonosisGPS library

The GeonosisGPS library is responsible for communicating with the GPS / GNSS hardware component, performing the following tasks:

- Configures the device using the appropriate configuration parameters. This is performed by sending the appropriate binary data to the virtual com port that is created by the USB connection and the GPS / GNSS board driver.
- Receives and parses NMEA sentences and raw data transmitted by the receiver. This is performed by establishing a connection to the same com port and parsing data transmitted by the receiver, in order to create GPGXX objects according to the NMEA specification.
- Sends RTCM corrections to the receive so as to achieve cm – level position accuracy by RTK solutions. This is accomplished by establishing a Berkley socket connection between the GeonoesisGPS library and NTRIP caster, by providing IP, port, and credentials to establish the connection.

### 3.1 Library description

The main class of the GeonoesisGPS library is a GPShandler object which allows asynchronous communication with GPS / GNSS component so as to enable user interaction with other components of the application that consumes the library, e.g., user interface with windows forms. GPShandler uses a Serial Port class that allows to read and write messages to the serial port. It also contains a NMEA namespace which provides classes for each one of the NMEA sentences transmitted by the GPS / GNSS device and is responsible to decode the NMEA sentences into a meaningful format. Finally, it provides a NTRIP namespace that handles connection to the NTRIP caster. The UML diagram of the GeonoesisGPS library can be found in Figure 2.



### 3.2 Namespace NMEA


The NMEA sentences that are decoded by our GeonoesisGPS library can be found in following Table 1. Each sentence corresponds to a class in our library.

NMEA Sentence	Sentence contents
\$GNGGA	<i>Time, position, and fix related data of the receiver.</i>
\$GNGLL	<i>Position, time and fix status.</i>
\$GNGSA	<i>Used to represent the ID's of satellites which are used for position fix.</i>
\$GNGST	<i>Provides System Fix Data such as solution status, STDevE, N, Z etc.</i>
\$GNGSV	<i>Satellite information about elevation, azimuth and CNR, \$GNGSV is used for GPS, Galileo and Beidou satellites</i>
\$GNRMC	<i>Time, date, position, course and speed data.</i>

*Table 1: Overview of Ublox's F9P NMEA sentences processed by GeonoesisGPS*








In the following we provide class description (properties / methods) for all classes in the NMEA namespace. Table 2 summarizes the annotation used in the rest of the document for specifying the type of Property / Method / Events for all following tables.

	Property
	Method

	Event
---	-------





**Table 2:** Annotation used for classes' properties and methods.

- **Class GPGSA:** Implements methods for decoding a GNGGA NMEA sentence, providing the respective information in the form of properties of the object. Properties and methods implemented are displayed in the following Table 3.

GPGSA (NMEASentence)		Constructor method that decodes the provided string and assigns property values
Mode		Mode. M=Manual, A=Auto (forced/not forced to operate in 2D or 3D mode)
FixMode		Enumeration for the GSA Fix mode. Takes values between FixNotAvailable, 2D Fix and 3D Fix
PRNInSolution		PRN Numbers used in solution
PDOP		Point Dilution of Precision
HDOP		Horizontal Dilution of Precision
VDOP		Vertical Dilution of Precision









**Table 3:** Properties and Methods of GPGGA Class


- **Class GPGLL:** Implements methods for decoding a GNGLL NMEA sentence, which is responsible for transmitting Geographic position, Latitude and Longitude, providing the respective information in the form of properties of the object. Properties and methods implemented are displayed in the following Table 4.

GPGLL (NMEASentence)		Constructor method that decodes the provided string and assigns property values
Position		Current position
TimeOfSolution		Time (UTC) Of Position Solution
DataValid		Data valid (true for valid or false for data invalid).

**Table 4:** Properties and Methods of GPGLL Class







- **Class GPGGGA:** Implements methods for decoding a GNGGA NMEA sentence, which is responsible for transmitting Time, position, and fix related data of the receiver, providing the respective information in the form of properties of the object. Properties and methods implemented are displayed in the following Table 5.

GPGGGA (NMEASentence)		Constructor method that decodes the provided string and assigns property values
TimeOfFix		Time of Fix (UTC)
Position		Coordinate of received position
FixQuality		Enumeration for the Fix Quality. Takes the following values: GPS, DGPS, Fix, Float, Estimated, Invalid
NoOfSats		Number of satellites being tracked.
Altitude		Altitude above sea level
AltitudeUnits		Altitude Units - M (meters).
Dilution		Horizontal dilution of position (HDOP).

HeightOfGeoid		Height of geoid (mean sea level) above WGS84 ellipsoid.
---------------	---	---






**Table 5:** Properties and Methods of GPGGA Class

- **Class GPGST:** Implements methods for decoding a GNGST NMEA sentence, which is responsible for transmitting System Fix Data such as solution status, STDevE, N, Z etc., providing the respective information in the form of properties of the object. Properties and methods implemented are displayed in the following Table 6.

GPGST (NMEASentence)		Constructor method that decodes the provided string and assigns property values
TimeOfFix		Time of Fix (UTC)
StDevE		Standard Deviation Easting
StDevN		Standard Deviation Northing
StDevEl		Standard Deviation Elevation
NMEAsentence		The NMEA sentence before decoding it

**Table 6:** Properties and Methods of GPGST Class

- **Class GPGSV:** Implements methods for decoding a GNGST NMEA sentence, which is responsible for transmitting System Initializes NMEA "Satellites in view", providing the respective information in the form of properties of the object. Properties and methods implemented are displayed in the following Table 7.









GPGST ()		Constructor that creates a new list of satellite
SatsInView		Number of satellites in view
Satellites		List of satellites. Satellite is a nested class containing satellite info: PRN Pseudo-random number ID Elevation above horizon (degrees) Azimuth (degrees) Signal-to-noise ratio in dBHZ (0-99)
GetSatelliteByPRN		Returns a satellite by its PRN
AddSentence (NMEASentence)		Adds a GPGSV sentence and parses it.

**Table 7:** Properties and Methods of GPGSV Class

### 3.3 Namespace NTRIP










The Namespace NTRIP is provides classes for establishing connection to a NTRIP caster that sends RTCM corrections to the GPS / GNSS receiver. In the following we provide class description (properties / methods) for all classes in the NTRIP namespace.

- **Class NTRIPNetwork:** Implements the network that provides the data streams to be consumed by the NTRIP client. Properties and methods implemented are displayed in the following Table 8.

ParseFromString (line)		Static constructor method that creates a new NTRIPNetwork from a SOURCETABLE string provided by a NTRIP server
Identifier		Network identifier, e.g., name of a network of GNSS permanent reference stations
Operator		Name of institution / agency / company operating the network
Authentication		Type of Authentication required. None, Basic, Digest
Fee		Specifies whether a user fee is required for receiving data streams from this network
WebAddress		Web-address for network information
WebStream		Web-address for stream information
WebRegistration		Web address or mail address for registration







**Table 8:** Properties and Methods of NTRIPNetwork Class

- Class NTRIPCaster:** Implements the NTRIP Caster that provides the data streams to be consumed by the NTRIP client. The NtripCaster is basically an HTTP server supporting a subset of HTTP request/response messages and adjusted to low bandwidth streaming data (from 50 up to 500 Bytes/sec per stream). Properties and methods implemented are displayed in the following Table 9.














ParseFromString (line)		Static constructor method that creates a new NTRIPCaster from a SOURCETABLE string provided by a NTRIP server
Host		Caster Internet host domain name or IP address
Identifier		Caster identifier, e.g. name of provider
Operator		Name of institution / agency / company operating the network
NMEA		Capability of Caster to receive NMEA message with approximate position from Client
Country		Three-character country code in ISO 3166
Latitude		Caster's position, latitude (north)
Longitude		Caster's position, longitude (east)
FallbackHost		Fallback Caster IP address

**Table 9:** Properties and Methods of NTRIPCaster Class

- Class NTRIPDataStream:** Implements the Data Streams that the NTRIP Server provides. Properties and methods implemented are displayed in the following Table 10.












ParseFromString (line)		Static constructor method that creates a new NTRIPNetwork from a string provided by a NTRIP caster
MountPoint		Caster mountpoint
Identifier		Source identifier, e.g. name of city next to source location
Format		Data format RTCM, RAW, etc.
FormatDetails		e.g., RTCM message types or RAW data format etc., update rates in parenthesis in seconds
Carrier		Data stream contains carrier phase information



		0 = No , 1 = Yes, L1 , 2 = Yes, L1 & L2
NavSystem		Navigation system(s)
NetWork		Network
Country		Three character country code in ISO 3166
Latitude		Position, latitude, north
Longitude		Position, longitude, east
NMEA		Necessity for Client to send NMEA message with approximate position to Caster (Boolean)
Solution		Stream generated from single reference station or from networked reference stations (0=single base, 1 = network solution)
Generator		Hardware - or software generating data stream
Compression		Compression algorithm (string)
Authentication		Access protection for this data stream, None, Basic, Digest
Fee		User fee required for receiving this data stream
BitRate		Bitrate of data stream, bits per second
Miscellaneous		Miscellaneous information

**Table 10:** Properties and Methods of NTRIPDataStream Class

- **Class SourceTable:** It contains the collection of DataStreams, NTRIPCasters and NTRIPNetworks provided by the NTRIP server.
- **Class NTRIPClient:** Provides the full functionality for accessing a NTRIP service. Properties and methods implemented are displayed in the following Table 11

Name		The NTRIP profile name
UserName		The NTRIP server Username
Password		The NTRIP server password
IP		The NTRIP server IP to connect to
Port		The NTRIP server port to connect to
MountPoint		The default MountPoint from which to request data stream
GPS		The GPSHandler object on which the NTRIPClient will send to RTCM corrections transmitted by NTRIP Server
NTRIPClient		Creates a new NTRIPClient object based on the provided values
GetSourceTable		Gets the SourceTable information provided by NTRIP Server, in the form of an object
StartNTRIP (MountPoint, GPSC)		Opens the connection to the NTRIP server and starts receiving data
StopNTRIP		Stops receiving data from the NTRIP server

**Table 11:** Properties and Methods of NTRIPClient Class













To use the above object classes, one has to create a new GPSHandler object that is responsible for connecting to the GPS / GNSS board, create a new NTRIPClient with the appropriate information (IP, port, credentials etc.), and all objects undertake their role to provide RTK solutions.



### 3.4 Namespace GeonoesisGPS








Namespace GeonoesisGPS contains two main classes which are responsible for the communication with GPS / GNSS device via com ports, i.e., the GPShandler and the Serial Port class. Both operate in an asynchronous way; moreover, GPShandler provides a way for asynchronous communication with a parent user interface (such as control, Windows Form, etc.) to be able to display results and statuses provided by the respective boards. In the following we provide class description (properties / methods) for both classes:

- **Class SerialPort:** Implements the network that provides the data streams to be consumed by the NTRIP client. Properties and methods implemented are displayed in the following Table 12.

SerialPort (SerialPort, BaudRate)		Initializes the serial port with values specified in argument
BaudRate		The com port's Baud rate
IsPortOpen		Boolean value indicating if port is opened
LogFile		The file name of where to write raw data received on port
Port		The name of the com port
Timeout		The port's timeout in seconds
Open		Opens the GPS port and starts parsing data
Start		Opens the serial port and starts parsing NMEA data. Returns when the port is closed.
Stop		Closes the port and ends the thread.
Write		Writes data to serial port. This is useful for sending RTCM data to the device.
WriteLog		Writes com log data
NewGPSData		Event that is fired where new data are available. Event is consumed by GPShandler class.

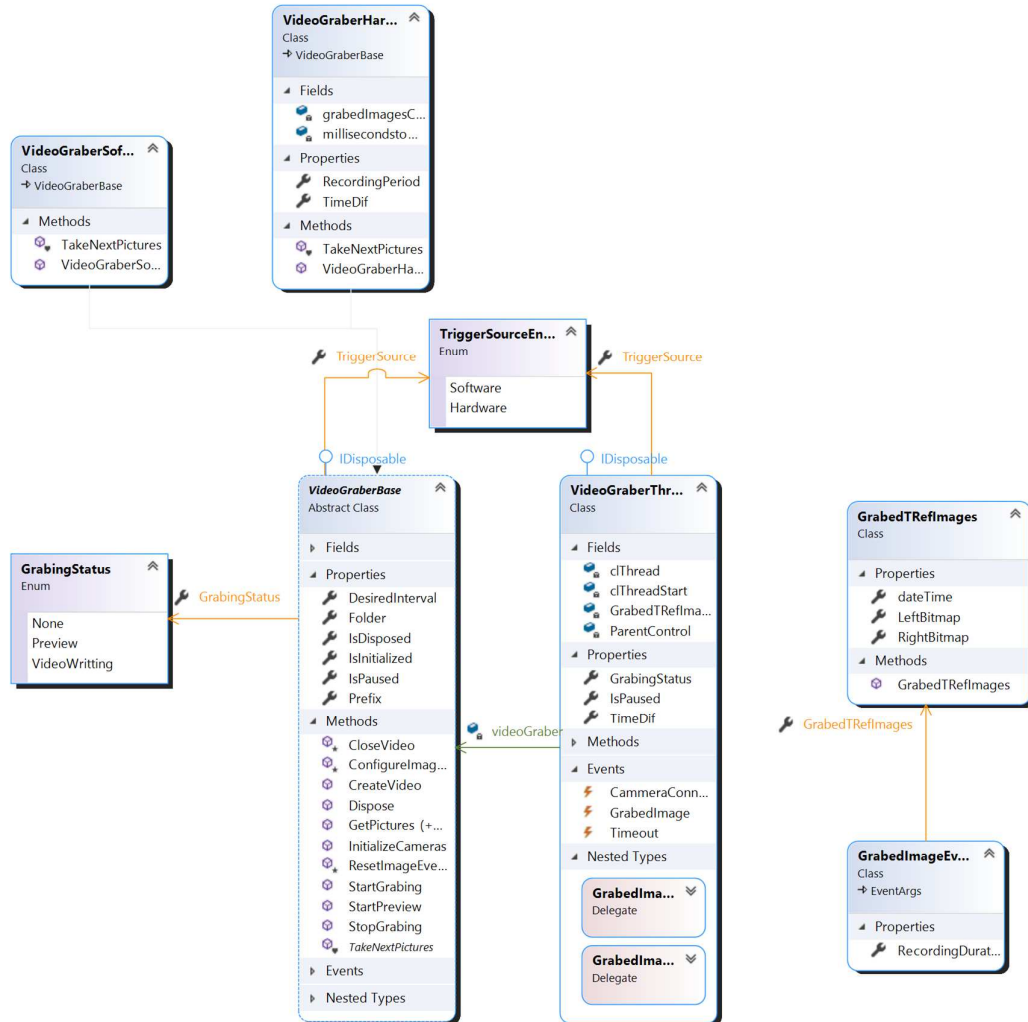
*Table 12: Properties and Methods of SerialPort Class*

- **Class GPShandler:** Implements the network that provides the data streams to be consumed by the NTRIP client. Properties and methods implemented are displayed in the following Table 12.

GPShandler (control)		Initializes the GPShandler with a parent control in argument which consumes the GPS data.
GPSFreq		Enumeration of the frequency on which GPS transmits data on the GPShandler and the hardware trigger (PPS). Takes values of 1 and 2 Hz.
HasGPSFix		The file name of where to write raw data received on port
IsPortOpen		Boolean value indicating if port is opened
LogFile		The file name of where to write raw data received on port
Timeout		The port's timeout in seconds
Configure		Configures the GPS / GNSS board according to the default configuration and the GPSFreq value

Start	⚙️	Opens the serial port and starts parsing NMEA data. Returns when the port is closed.
Stop	⚙️	Closes the port and ends the thread.
WriteToGPS	⚙️	Writes data to the GPS device. For instance, RTCM data for Differential GPS.
NewGPSData	⚡	Event that is fired when new data are available. The event broadcasts GPRMS, GPGGA, GPGSA, GPRMC, GPGSV, GPGST objects.

*Table 13: Properties and Methods of GPShandler Class*



*Figure 3: Class (UML) Diagram of VideoGrabber library*

## 4 VideoGrabber library

VideoGrabber library is written in C# consuming methods provided by FLIR's API regarding our imaging subsystem, i.e., SpinnakerNET responsible for communicating and configuring the cameras, and SpinVideoNET, responsible for writing avi video files. Files created by VideoGrabber classes use the MJPG format, that is, without MPEG compression, allowing us save high quality image data.












### 4.1 Library description





Library provides mainly a Video Grabber object that is responsible for connecting to our imaging subsystem, writing video files, and allowing to communicate asynchronously with a parent user interface object, such as a windows Form. This is achieved via multithreaded programming so as to be able to display results and statuses provided by the imaging subsystem without interrupting user experience in the user interface.

### 4.2 VideoGrabber Classes

The following classes are available by the VideoGrabber library. The respective UML diagram is illustrated in Figure 3




- Class VideoGrabberBase:** This is an *abstract* class that implements most properties and methods used in the image grabber. Implementations of this abstract class are the VideoGrabberSoftware and VideoGrabberHardware which specifically implement video grabber classes with software and hardware triggers, respectively. Properties and methods implemented in the VideoGrabberBase class are displayed in the following Table 14.

CreateVideo		Initializes a new avi file to be filled with video data
GetPictures (writeVideo)		Gets pictures and writes them to the video files if indicated by writeVideo argument
InitializeCameras		Connects to the cameras. Returns true id successfully connected
StartGrabbing		Starts grabbing images appending them to video file. If it is already writing video it does nothing
StartPreview		Starts grabbing images without appending them to video file
StopGrabbing		Stops grabbing images
DesiredInterval		The desired interval between grabbed images. This is only the desired interval and not the actual interval that images are collected, due to possible inability of the underlying hardware to service the request. This property is meaningful only where trigger is set to Software.
Folder		The folder on which the video file will be created
IsDisposed		A Boolean value indicating whether object is disposed
IsInitialized		A Boolean value indicating whether cameras are initialized
IsPaused		A Boolean value indicating whether grabbed images are currently being append to the video file

Prefix		The prefix of the name of the video (avi) files created
TriggerSource		Enumerator indicating the source of the camera trigger. Possible values are <i>Software, Hardware</i>
GrabbingStatus		Enumerator indicating the source of the camera trigger. Possible values are <i>Software, Hardware</i>
GrabedImage		Event that is fired when a new image has been collected by the hardware








**Table 14:** Properties and Methods of VideoGrabberBase Class






- **Class VideoGrabberSoftware:** Implements a video grabber object with a software trigger (computer controls image triggering)
- **Class VideoGrabberHardware:** Implements a video grabber object with a hardware trigger provided by the GPS / GNSS board through GPIO cables. Properties and methods implemented additionally to the ones inherited by VideoGrabberBase, are displayed in the following Table 15.

VideoGraberHardware		Creates a new object with hardware trigger
RecordingPeriod		The temporal period that the grabber operates recording images in the avi files
TimeDif		The time difference between GPS time and time of host computer

**Table 15:** Properties and Methods of VideoGrabberHardware Class

- **Class GrabedTRefImages:** Class that is used to pass stereo images grabbed by our stereo rig, i.e., Left and Right bitmaps, along with the time the collection happened.
- **Class VideoGrabberThread:** Implements a thread that asynchronously grabs images through a Software or Hardware Video Grabber object. Class is used to offer a seamless user experience, allowing to update user interface components through multithreading. Properties and methods implemented are displayed in the following Table 16.

VideoGraberThread (control, trigger)		Creates a new video grabber thread object with the trigger specified in argument
VideoGraber		The underlying VideoGrabberBase (software or hardware object)
GrabingStatus		The temporal period that the grabber operates recording images in the avi files
TimeDif		The time difference between GPS time and time of host computer
IsPaused		A Boolean value indicating whether grabbed images are currently being append to the video file
GetPictures		Returns the next pictures that are grabbed by the hardware in the form of a GrabedTRefImages object
SetTimeDiff		Method that sets the TimeDif value. Usually, the maximum difference between GPS and Host time during the last, e.g., 100 events.

StartPreview		Starts cameras, firing GrabedImage events, without actually writing a video avi file
StartVideo		Starts cameras, firing GrabedImage events, writing a video avi file
Stop		Stops cameras and releases all resources
CameraConnected		Event that is fired when cameras are successfully connected to the process and have starts image grabbing
GrabedImage		Event that is fired when a new image has been collected by the hardware

**Table 16:** Properties and Methods of VideoGrabberThread Class

## 5 Map Matching and Routing

### 5.1 Map Matching Algorithm

Map matching is the problem of how to match recorded geographic coordinates to a logical model of the real world, typically using some form of Geographic Information System. The most common approach is to take recorded, serial location points (e.g., from GPS) and relate them to edges in an existing street graph (network), usually in a sorted list representing the travel of a user or vehicle. Routing quality and performance.

In our case the Map Matching Algorithm should be used in order to determine the network road segments (graph edges) that have already been visited. A naïve approach in map matching would be to simply determine the edge that is closer to each sampled GPS point; however, this approach would result in extremely unreasonable paths involving strange U-turns, inefficient looping, and overall bizarre driving behavior.

The key problem thus in map matching is the tradeoff between the roads suggested by the location data and the feasibility of the path. To avoid unreasonable paths, we can introduce knowledge of the connectivity of the road network to help pull the solution away from clearly bizarre behavior [5]. In [6] a novel, principled map matching algorithm is described that uses a Hidden Markov Model (HMM) to find the most likely road route represented by a time-stamped sequence of latitude/longitude pairs. The HMM elegantly accounts for measurement noise and the layout of the road network.

The HMM models processes over a path through many possible states, where some state transitions are more likely than others and where the state measurements are uncertain. In the algorithm of [6], the states of the HMM are the individual road segments, and the state measurements are the noisy vehicle location measurements. The goal is to match each location measurement with the proper road segment. This state representation naturally fits the HMM, because transitions between road segments are governed by the connectivity of the road network.

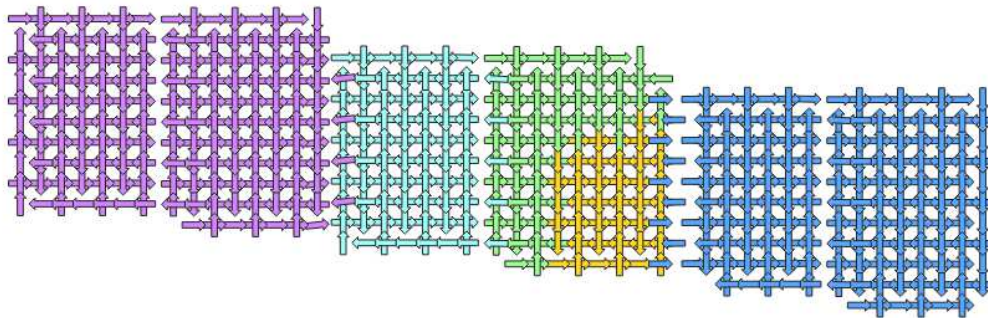
The algorithm of [6], is implemented in barefoot project developed in Java [7], an open-source Java library for online and offline map matching. Barefoot is ported from Java to .NET via [8] which is used as basis for our implementation.

Specifically, we have used [8] and adjusted it into our context using the classes presented in [D12] and [D13] (vertices derived from Geoapi.Coordinate and Edges connecting two

vertices. We have tested the performance of the implementation and we have established that it performs properly on the kind of GPS data collected by our system.

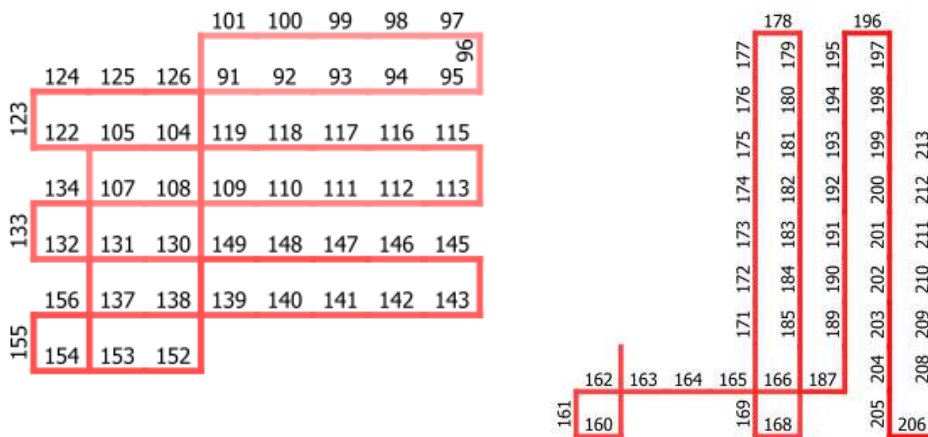
## 5.2 Routing algorithm performance

Given that the routing algorithm of [D12] should be run in a real time environment we run a series of experiments in order to check its performance and confirm its ability to be run under such circumstances. We used several synthetic road network data constituted by of 70, 170, 307, 604 and 882 road segments, representing edges of the respective graph (Figure 4). Figure 5 displays a part of the solution for the network of 170 edges (green and yellow edges of Figure 4).



**Figure 4:** Synthetic road network. Different edge colours indicate different group of edges

We run the experiments on a i7-4510U @ 2GHz notebook and we acquired the results illustrated in Figure 6 regarding the algorithm’s execution time.

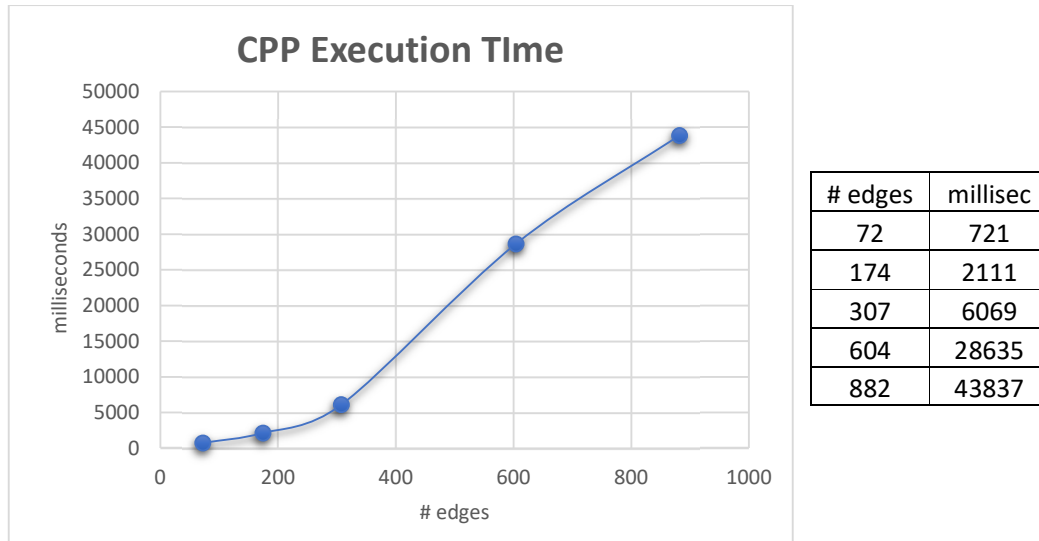


**Figure 5:** Part of a CPP example solution (segments 91 to 213)

It is clear that the performance of the algorithm downgrades as the number of network edges increases. For a network of e.g., 800 road segments, we have an execution time



greater than 40 seconds. Moreover, we have experimentally established that in a day's work MOBILO system could gather data from approximately 160 km of road data. Given a mean segment length of 100 m (between consecutive road junctions in an urban area), a typical network to be covered in one day's work, should contain about 1600 segments. Therefore, the algorithm's execution time in such conditions would exceed one minute making it clearly unable to support real-time environments.

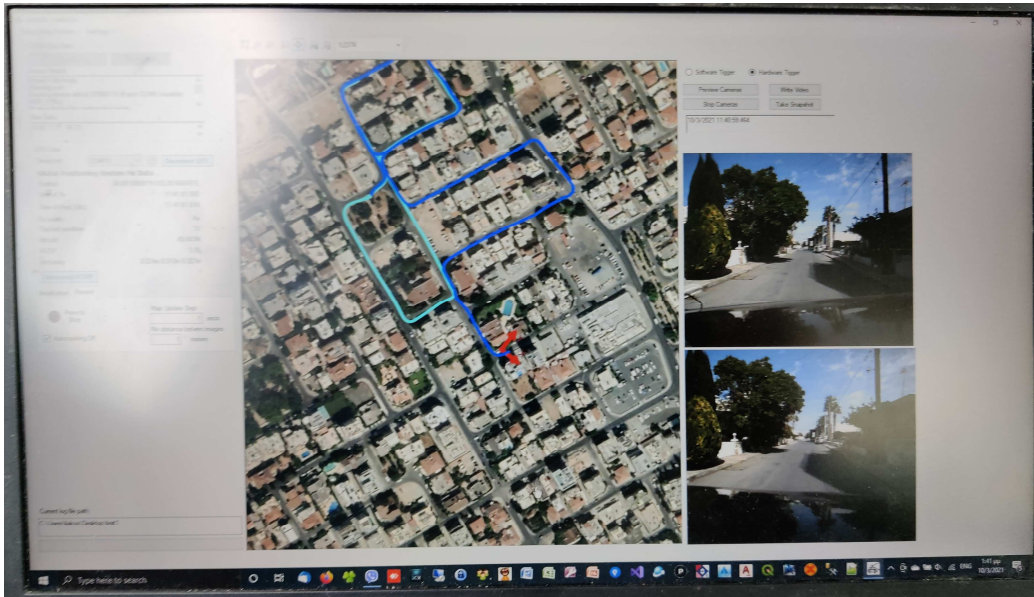


**Figure 6:** Chinese Postman algorithm execution time vs number of edges in the graph

Moreover, during data collection the vehicle would not always follow the directions given by the algorithm, causing the algorithm to be continuously run after each root modification. What is more, given that all the tasks that are needed to be performed by the data collection software, e.g., manage GPS / GNSS, INS, receiver images from cameras, saving to disk etc., it would be a mistake to charge CPU with such a demanding task.

## 6 MobiloGrabber Application

MobiloGrabber application is the main output of WP6. It is a windows application used to collect data in the field, controls all components connected to a laptop with windows 10 and a wireless connection to the internet. MobiloGrabber software assures the synchronization between all connected modules, that is, the F9P board (integrated into GPS-RTK2), MTi-7 INS/GNSS and FLIR (machine vision) cameras. For this purpose, it consumes the GeonosisGPS and VideoGrabber libraries presented in the previous sections and utilizes their asynchronous capabilities in order to provide a user-friendly interface. An example screenshot of MobiloGrabber application during in-road data collection can be found in the following Figure 8.



*Figure 7: Example screen of MobiloGrabber software during data collection*

Additionally, MobiloGrabber application also consumes XSens's API in the form of XDA classes provided as source code that must be integrated into the application's source code; the respective API provides classes that can be used to access (connect, configure, read from) the MTi-7 application board.

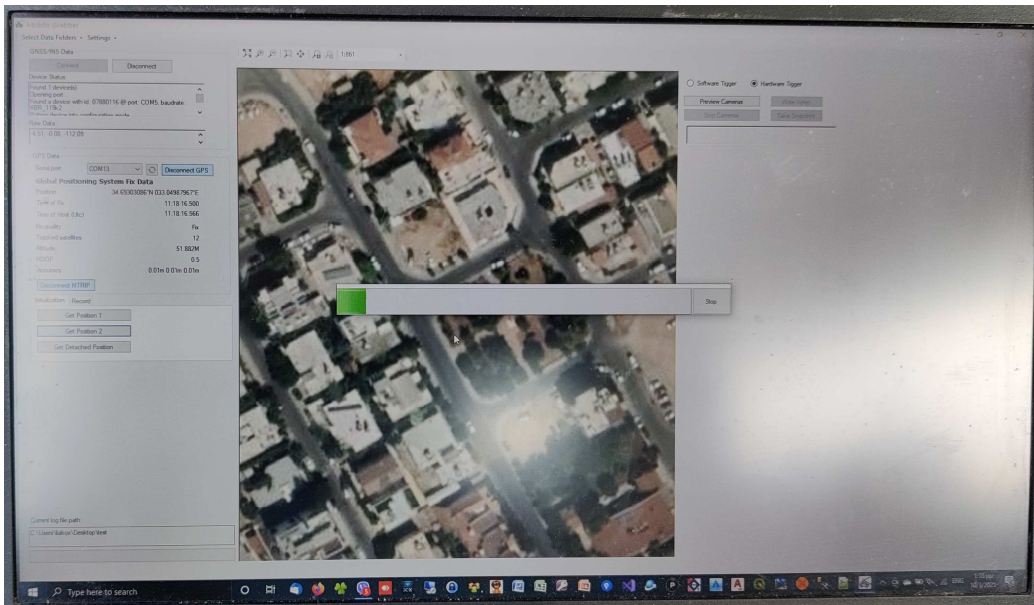
### 6.1 General Description

Given the data provided by the connected sensors of the positioning subsystem (current position and heading), MobiloGrabber is capable to display the system's current position on a map, as well as all the recorded so far trajectory of the respective vehicle. The software also displays several frames of the gathered video data so as to give the user the ability to check whether all components are working well. Summarizing, the following indications are displayed over the UI of our software

- GPS Condition (fix / float / autonomous / timeout)
- MTi-7 connection status (connected / timeout)
- Machine vision Cameras (gathered frames are displayed every, e.g., 5 sec)

Regarding the maps used in the background we have exploited the developments of [D13]. MobiloGrabber is capable of displaying raster maps, as well online XYZ tile – based maps, such as google maps, Bing maps, OpenStreetMap etc. It also supports raster and vector data sources that can be found in the local computer. As such, the user has the ability to choose among several backgrounds which are suitable for her needs. The software also supports several projection systems for the geographically displayed data, including WGS84, UTM, Cyprus's LTM etc. On the other hand, all data collected are stored in the application's data folder in the underlying system on which data are collected, i.e., WGS84.





*Figure 8: Example screen of MobiloGrabber software during system's initialization*

All data acquisitions are performed in an asynchronous manner, where the main UI component is regularly refreshed, while the user experiences no delays in the tasks requested, i.e., zooming – panning the map etc. An example screen of the UI that assist the video collection process is displayed in the following Figure 7.

MobiloGrabber application provides the following capabilities to the end – user:

- Connect to the INS device and display its status / measures taken
- Connect to the GPS / GNSS receiver through com port. Configure GPS / GNSS receiver during connection.
- Allow to change the com port on which to scan for the device
- Connect to NTRIP servers and broadcast RTCM corrections to the GPS / GNSS receiver. Manage profiles of multiple NTRIP servers to be used.
- Connect to the imaging subsystem. Select software or hardware trigger for the cameras. Software trigger is used in e.g., lab conditions where no connection to the GPS / GNSS system is possible.
- Initialize MOBILO system by storing initial GPS positions in several epochs ( Figure 8)
- Select folder on which video / trajectory data are stored
- Add backgrounds in the form of XYZ tiles, raster and vector maps
- Display current and previous trajectories over the selected base map
- Display current vehicle's position and heading over the selected base map
- Display currently taken images by the imaging subsystem
- Control storing parameters: minimum distance between images

When using network data (shapefiles), the software is capable of map-matching the current trajectory over them so as to mark already visited road segments and help the user to decide over his / hers next trajectory, i.e., where to move next. As already stated, on-line

running of the algorithm that fully covers the network is prohibited by current status of existing hardware due to performance issues.

We have tested the performance of the developed software on several configurations. Among the three collecting processes obviously the most demanding in terms of CPU utilization, RAM and storage utilization, is the process that captures imagery data in the form of Video files. We have currently established that a rather old i7-4510U @ 2GHz notebook, with at least three USB built in interfaces, is capable of serving the developed app and work seamlessly during a data collection procedure of several hours.

## 6.2 Produced files

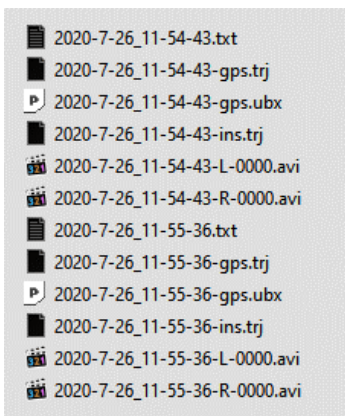
MobiloGrabber produces *projects* in the form of folders on which all recorded files are stored. Files names are structured by their starting time which serves as an identifier. Their exact structure is presented in Table 17. Regarding the notation used, [DateInfo] denotes a string in the form of YYYY-MM-dd\_HH-mm-ss, e.g., 2021-03-31\_09-30-00, while [NNNN] denotes a serial number starting spanning from 0 to 9999, i.e., 0000, 0001, 0002 etc. Each folder should contain files collected at several time periods, *as long as the system setup has not been changed*. For example, data collected in the period of one day, regardless of any stops happened, should be stored in one folder, as long as MOBILO's carrier and GPS position has not been changed. The [NNNN] part of the files is used since video files with size greater 2GB are split across several files.

The following Table 17 contains a description about all files gathered during a MOBILO collection process. MOBILO's project may contain data from several such collections. Therefore, a project folder may contain several file groups that comply with the description of Table 17; an example of these data groups is illustrated in Figure 9 displaying the contents of a MOBILO project that is composed by two separate collection processes, i.e., collection process that started at 2020-07-26 11:54:43, and collection process that started at 2020-07-26 11:55:36.

Filename	Description
[DateInfo].txt	Contains date/time information regarding the video that was started at the [DateInfo] timestamp.
[DateInfo]-L-[NNNN].avi	The <i>n</i> 'th video file of the left camera that was started at the [DateInfo] timestamp. It should be noted that FLIR's API that is used to produce avi files divides collected video files up to a size of 2GB. For example, in the case where a video collection procedure lasts several hours without stopping it, thus producing avi files up to 10 GB, 5 files will be produced denoted with the name [DateInfo]-L-0000.avi, [DateInfo]-L-0001, ..., [DateInfo]-L-0005.
[DateInfo]-R-[NNNN].avi	Same as previous regarding the right camera.
[DateInfo]-gps.trj	Contains trajectory information obtained by GPS/GNSS regarding the data collection process that was started at timestamp denoted by [DateInfo]. Specifically, it contains GPS/GNSS position measurements obtained by Ublox F9P, along with datetime, fix status and achieved precisions. Data are logged in WGS84.
[DateInfo]-gps.ubx	Contains raw data collected by GPS/GNSS F9P regarding the data collection process that was started at timestamp denoted by [DateInfo].

Filename	Description
[DateInfo]-ins.txt	Contains trajectory information obtained by INS/IMU regarding the data collection process that was started at timestamp denoted by [DateInfo]. Specifically, it contains measurements obtained by MTi-7. Data are logged in WGS84.
[DateInfo]-cmdGetPosition1.txt	Contains measurement information about the initial position obtained by our GPS / GNSS in MOBILO's system position 1
[DateInfo]-cmdGetPosition2.txt	Contains measurement information about the initial position obtained by our GPS / GNSS in MOBILO's system position 2
[DateInfo]-cmdGetDetachedPosition.txt	Contains measurement information about the initial position obtained by our GPS / GNSS in MOBILO's system position 1

*Table 17: Project files containing actual on-site data.*



*Figure 9: Example data contained in a MOBILO project folder.*

## 7 Conclusions

This deliverable describes part of the results of MOBILO's project WP6 which primarily aims at developing a software component that is used by the system's end user before and during the data collection process. Specifically, as suggested by the project proposal, the component should at least provide the following functionality (a) Display a map, (b) Gather Video Data (c) Gather GPS / INS Data. This software component is very essential to the end user to assist her during the – demanding in terms of focalization - video collection process: record and mark the already visited road network parts, while at the same time supervising the video collection process and adjusting possible interface parameters. The software developed by this WP can be used to pair with the MOBILO hardware components, such as GPS RTK data, INS and Cameras. Specifically, the data gathering developed software, utilizes all components connected to a laptop with windows 10 and a wireless connection to the internet. The software assures the synchronization between all connected modules, that is, the GPS-RTK2 board, MTi-7 INS/GNSS and FLIR (machine vision) cameras.

Regarding the demonstrated inability to on-line process the Chinese Postman Algorithm due to performance issues, a possible future solution would be to implement a server-side service that could process the demanding algorithm without the limitations posed by the on-site hardware usage.

During this WP we have developed two underlying libraries (APIs) that support this interface. This option was dictated by the need for code reuse, as well as the possible universal usages of several components developed during the project's execution. For example, *GeonoesisGPS* library developed under this work package, is used in our MobiloGrabber application, while it can be used for any GPS / GNSS related application that exploits u-blox F9P board [1]; in fact, we have already developed a proof-of-concept application that uses F9P board and *GeonoesisGPS* library to perform simple topographic measurements, a by-product of our project.

## 8 References

- [1] Ublox, "ZED-F9P," UBlox, [Online]. Available: <https://www.u-blox.com/en/product/zed-f9p-module>.
- [2] "NetTopologySuite," [Online]. Available: <https://nettopologysuite.github.io/>. [Accessed 30 9 2020].
- [3] "SharpMap," [Online]. Available: <https://github.com/SharpMap>. [Accessed 20 9 2020].
- [4] "SpinView," [Online]. Available: <http://softwareservices.ptgrey.com/Spinnaker/latest/page4.html>. [Accessed 31 03 2019].
- [5] S. Brakatsoulas, D. Pfoser, R. Salas and C. Wenk, "On Map-Matching Vehicle Tracking Data," in *VLDB*, Trondheim, 2005.
- [6] P. Newson and J. Krumm, "Hidden Markov Map Matching Through Noise and Sparseness," in *ACM GIS*, Seattle, 2009.
- [7] "Github/bmwcarit/barefoot," [Online]. Available: <https://github.com/bmwcarit/barefoot>.
- [8] "Sandwych.MapMatchingKit," [Online]. Available: <https://www.nuget.org/packages/Sandwych.MapMatchingKit/>.