SMART and FLEXible mobile DATA COLLECTOR for GIS

(Acronym, MOBILO)

ENTERPRISES 0916/0055

[D10] Report in WP4 activities

(Mobile System Positioning)

Deliverable n.	D10	**Deliverable title**	Report on WP4 Activities
Workpackage	WP4	**WP title**	Mobile System Positioning
Editors	Elias Frentzos (GEO), Dimitrios Skarlatos (CUT)		
Contributors	Elias Frentzos (GEO), Dimitrios Skarlatos (CUT), Lefteris Tournas (CUT)		
Status	Final		
Distribution	Confidential		
Issue date	2019-11-29	**Creation date**	2019-11-01

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
SDK	Software Development Kit
GPS/GNSS	Global Positioning System / Global Navigation Satellite System
INS/IMU	inertial navigation system / inertial measurement unit
RTK	Real Time Kinematic
NMEA	National Marine Electronics Association
RTCM	Radio Technical Commission for Maritime Services
NTRIP	Networked Transport of RTCM via Internet Protocol

REVISION CHART AND HISTORY LOG

REV	DATE	REASON
0.1	01/11/2019	Initial
0.2	15/11/2019	Draft
0.3	29/11/2019	Final

Executive Summary

Aim of this report is to demonstrate the positioning methods integrated into of MOBILO's system. Specifically, WP4 is responsible for the positioning subsystem of our system which also includes the calculation of the exterior orientation based on INS and / or other methods presented in this report. The tasks of this WP include the *Decision of the GPS and INS specifications*, investing into U-blox's high precision F9P board with accuracy of 0.01 m+ 1 ppm CEP horizontally and 0.01m +1 ppm CEP vertically and a low-cost INS which measures data rotation data with an RMS of Roll/pitch (static – dynamic) 0.5° and Yaw (dynamic) 1.5°. The *Setup of our system* is achieved by developing several software components that communicate with the devices, configure them, send to it RTCM messages provided by a NTRIP caster, read position solutions provided by the boards and store raw data. Physical communication with the boards is performed via USB interfaces while software communication is based on the NMEA specification and the provided SDKs using C# and .NET framework on Windows PC. We further tested existing open-source solutions, i.e., RTK Lib for post processing of obtained data; the comparison between all alternative shows that RTK solutions provided by U-blox F9P board match the solutions provided by standard expensive GPS / GNSS, while post processing with RTK Lib does not achieve acceptable results that can be used in our system. Finally, we introduce a method for obtaining accurate heading calculations for the MOBILO system based on arc approximations for the vehicle's movement and a delta heading calibration technique that calculates the system's heading relatively to the one of the vehicle, which significantly improves relative vectors provided by the INS.

1 Introduction

Aim of this report is to demonstrate the mobile positioning system employed in the MOBILO's final system. According to the submitted proposal, *MOBILO proposes a low-cost mobile mapping system which consists of a GPS / GNSS RTK, an inertial INS / IMU system which gathers position and orientation data, as well as video cameras to collect image data*. More specifically, this project aims at developing two low-cost alternative solutions (a) one with low-cost cameras (e.g., action cams) together with any existing RTK GPS, an alternative which reduces the cost of employed hardware to several hundreds of €, and targets to a specific customer group i.e., professional surveyors, and (b) high-end machine vision mission cameras together with RTK GPS / GNSS, INS / IMU which targets to more advanced users. These solutions will be enhanced with tools automating both the collection in the field, as well as the processing in office procedure. The mobile mapping system could me mounted on a variety of mobile platforms. The cameras, IMU and GPS antenna are required to be in a stable position throughout the data collection.

Regarding the specific tasks of WP4, these are closely related to the tasks of WP3, focusing on the continuous calculation of the accurate exterior orientation of the total system, which will be used to provide the respective exterior orientation of the imaging subsystem by adding the misalignment between the cameras and the mapping reference system calculated in WP3.

According to the Annex I of the contract, three tasks were to be performed during this WP execution, namely:

- A4.1: Decision of the GPS and INS specifications
- A4.2: Setup and trials for experimental data gathering
- A4.3: Test existing open-source Software libraries for post processing.

Along with the tasks that where predefined in the project's Annex I, a set of other tasks arouse during the project's implementation, namely

- Improving heading calculation, and,
- Testing the solutions provided by several alternatives

All these tasks will be presented in the following paragraphs.

2 Decision of the GPS and INS specifications

In 2019, U-Blox released a new high-performance dual frequency GNSS chip on the market, able to provide centimeter level accuracy at 150 € (U-Blox, 2020) while the respective boards / development kits on the market based on this chip do not exceed 300 €. This is a significant improvement that is incorporated in the developed system, in order to provide high positional accuracy without using expensive GPS equipment.

2.1 Positioning Subsystem

According to what has been described in the proposal, we have initially used a Geomax Zenith 25 GNSS paired with the low-cost system. The main reason towards this decision was the ability to directly record the NMEA results of the respective GNSS receiver. Recent advances however in the field of GNSS receivers / boards, directed us towards the employment of a low cost RTK board: while commercial GNSS / GPS receivers still cost several thousand euro, recently available board in the market are in the order of hundreds of euro, making it therefore easy to integrate them into our system.

Thus, we have tried U-blox F9P board integrated into a Sparkfun's GPS-RTK2 board. This is a very cost-effective solution since the board's price does not excess 300 € and is based on the recently announced F9P RTK board from U-blox. The main advantage of GPS-RTK2 board is that it can accept RTCM messages thus achieve fixed RTK solutions with accuracy of 0.01 m+ 1 ppm CEP horizontally and 0.01m +1 ppm CEP vertically, and also a typical convergence time in the RTK mode of less than 10 sec.

Parameter	Speci?cation	
Receivertype	Multi-band GNSS high precision receiver	
Accuracy of timepulse signal	RMS 99%	30 ns 60 ns
Frequency of timepulse signal		0.25 Hz to 10 MHz (con?gurable)
Operational limits[1]	Dynamics Altitude Velocity	? 4 g 50,000 m 500 m/s
Velocityaccuracy[2]		0.05 m/s
Dynamic heading accuracy[2]		0.3 deg

GNSS		GPS+GLO+GAL +BDS	GPS+GLO+GAL	GPS+GAL	GPS+GLO	GPS+BDS	GPS
Acquisition[3]	Cold start	24 s	25 s	29 s	26 s	28 s	29 s
	Hot start	2 s	2 s	2 s	2 s	2 s	2 s
	Aided start[4]	2 s	2 s	2 s	2 s	2 s	2 s
Nav. update rate	RTK	8 Hz	10 Hz	15 Hz	15 Hz	15 Hz	20 Hz
	PVT	10 Hz	12 Hz	20 Hz	25 Hz	25 Hz	25 Hz
	RAW	20 Hz	20 Hz	25 Hz	25 Hz	25 Hz	25 Hz
Convergence time[5]	RTK	< 10 s	< 10 s	< 10 s	< 10 s	< 10 s	< 30 s

Figure 1: U-blox F9P board integrated into Sparkfun's GPS RTK2, and specifications

2.2 Inertial Navigation Subsystem

We also obtained INS XSENS MTI-7-DK board which measures data rotation data with an RMS: Roll/pitch (static – dynamic) 0.5°, Yaw (dynamic) 1.5°. We concluded to this board mainly due to our aim to maintain hardware cost of our system's part in rather low levels. Currently there are also other alternatives provided by XSens and other manufacturers such as VectoNav; however, all these alternatives come with much higher cost, e.g., five time higher or more.

Orientation accuracy	
Roll/pitch (static)	0.5° 1σ RMS
Roll/pitch (dynamic)	0.5° 1σ RMS
Yaw (dynamic)	1.5° 1σ RMS
Position and velocity	
Horizontal position 1σ STD (SBAS)	1.0 m
Vertical position 1σ STD (SBAS, baro)	2.0 m
Velocity 1σ RMS	0.05 m/s

Figure 2: Xsens MTi-7 integrated into Development Kit, and specifications

According to the board's specification the provided orientation accuracy applied on the exterior orientation of our system, would result to a typical error of approximately

$$E_{(typical)} = S * \sqrt{\tan{(0.5)}^2 + \tan{(0.5)}^2 + \tan{(1.5)}^2},$$

where *S* is the distance of the actual object from the stereo rig, resulted to a typical error of approximately 60 cm in a distance of 20 m.

Moreover, according to the manufacturer MTi-7 board may suffer in the calculation of heading / yaw resulting on non-stable or incorrect measures when the heading observability is too low. This can occur when:

- the application is moving in a straight line
- the application is not moving or very slowly moving (less than 7 m/s), which is equal to 25 km/h
- GNSS signal is lost

According to this information it becomes obvious that the yaw measurements obtained by MTi-7 are highly depended on the GNSS solution provided by the daughter GNSS board that is employed in the MTi-7 (which is also a Ublox board).

On the other hand, given that in our system a high performance GNSS board capable of providing cm level GNSS solutions is employed, we may use it to calculate the heading and consequently the imaging subsystem's exterior orientation. Specifically, given a case where the GNSS board provides fixed solutions in the order of e.g., 3 cm, and a rather low speed of 3 m/sec, the typical error in the calculation of heading from consecutive fixed GNSS positions, would not exceed 0.6^o. In this report we investigate on this idea, significantly improving heading calculations, thus improving the accuracy of our system's provided solutions (see Section 5).

2.3 Physical Integration

We also used a Tallysman antenna capable of receiving L1/L2 signals from GPS, GLONASS and Galileo L5, with an additional cost of approximately 300 €. Alternative antennas at even lower costs are currently available by e.g., U-Blox [1], which have been also tested and found that performs accordingly.

(a) (b)

Figure 3: Tallysman and UBlox high precision antennas

The antenna signal was split using an appropriate signal splitter in order to direct the signal to both GPS-RTK2 and XSENS INS MTI-7-DK boards.

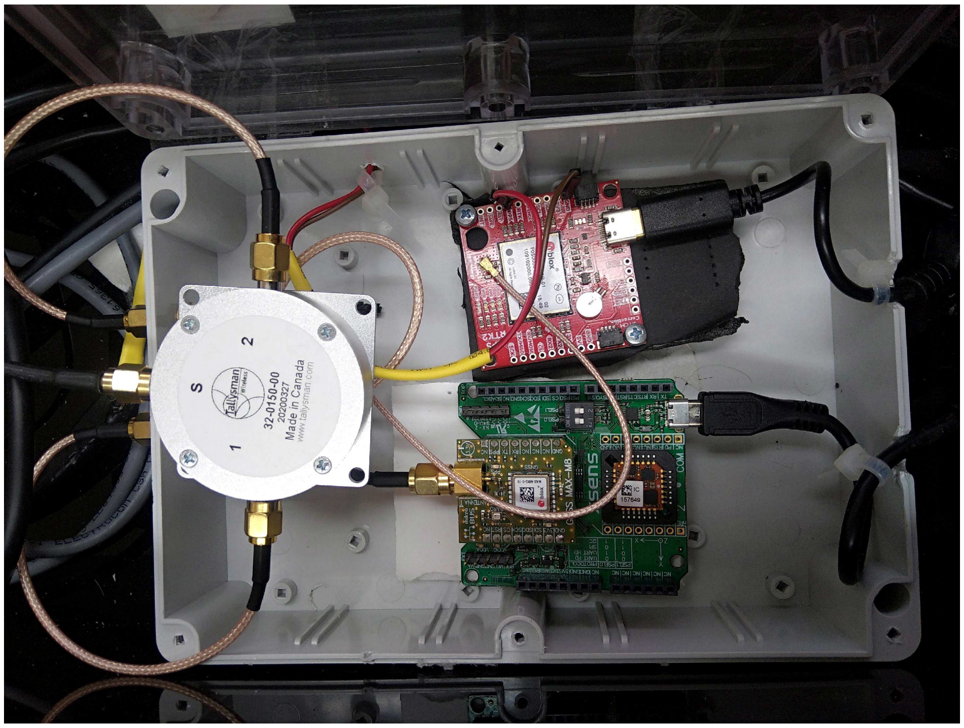


Figure 4: Project box for positioning sub-system

Both boards and antenna signal splitter were integrated into a “project box” (Figure 4) so as to stay protected from external environment conditions (humidity etc.), and provide single – point connectivity to our system: all parts were intra connected in the project box, while only two usb-female adapters and a SMA female adapter are leaving the project box.

3 Setup and trials for experimental data gathering

We have initially used U-blox software (U-center) [2] to establish communication with the board, as well as to get familiar with the configuration required. We have followed several approaches proposed by [2] to establish communication with GPS RTK2 and transmit RTCM data to it. Specifically, we used RTK Lib [3] and its component RTKNAVI to establish communication from the PC to the board and transmit RTCM data to it: all current proposals suggest connecting the board to the PC running RTKNAVI via two separate com ports. The first com port is used to receive NMEA data from the GPS RTK2 board, while the second is used to transmit RTCM corrections to it. The respective data flow is described in the following diagram (Figure 5).

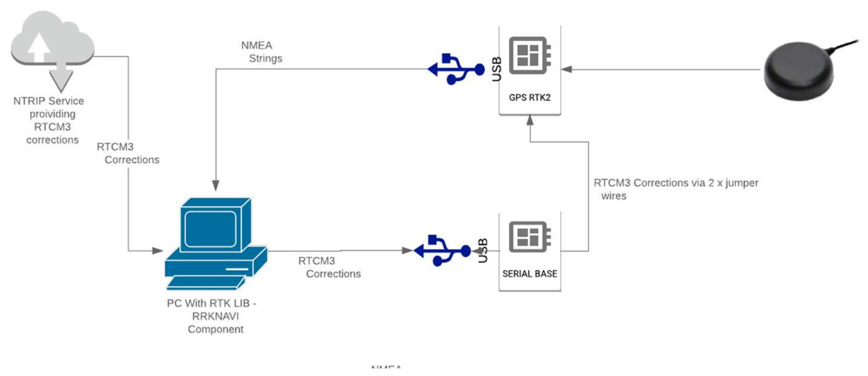

NMEA Strings
USB
GPS RTK2
NTRIP Service proividing RTCM3 corrections
RTCM3 Corrections
RTCM3 Corrections via 2 x jumper wires
RTCM3 Corrections
USB
SERIAL BASE
PC With RTK LIB - RRKNAVI Component

Figure 5: Initial data flow

After running our initial tests with the existing suggestions and provided software, we have further implemented a software component that directly communicates with GPS-RTK2 board via one interface (serial port) serving as both receiver and transmitter (Figure 6).

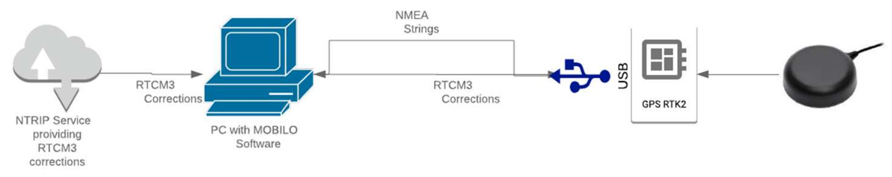


Figure 6: Final data flow

The developed software performs the following tasks:

- Configures the device using the appropriate
- Receives and parses NMEA data
- Sends RTCM corrections

which are further explained in the paragraphs to follow.

3.1 Configuring the device

The configuration is performed by sending several messages to the RTK board via the USB interface (virtual com port). Specifically, we configure the device so as to receive RTCM3 corrections via the serial interface, and also broadcast the respective solutions via the serial interface (NMEA sentences), as well as the raw GNSS measurement and navigation data. We also configured the device to broadcast solutions in a rate of 2 Hz, i.e., 2 solutions per second. The F9P board supports providing RTK solutions in a maximum rate of 8 Hz, however we chose to configure it in a lower rate so as to achieve lower communication rates and reduce the desirable computational cost (CPU, HDD etc.).

The sentences used to configure the device, along with their explanation are summarized in the following table:

Serial Sentence	Explanation
`0xB5, 0x62, 0x06, 0x01, 0x08, 0x00, 0x02, 0x15, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x27, 0x49`	*RXM-RAWX-USB: Transmittal of Multi-GNSS Raw Measurement Data for creating Rinex3 files through serial port*
`0xB5, 0x62, 0x06, 0x01, 0x08, 0x00, 0x02, 0x13, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x25, 0x3B`	*RXM-SFRBX-USB: Transmittal of Navigation data through serial port*
`0xB5, 0x62, 0x06, 0x24, 0x24, 0x00, 0xFF, 0xFF, 0x00, 0x03, 0x00, 0x00, 0x00, 0x00, 0x10, 0x27, 0x00, 0x00, 0x0F, 0x00, 0xFA, 0x00, 0xFA, 0x00, 0x64, 0x00, 0x5E, 0x01, 0x00, 0x3, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x88, 0x2`	*Cut-off angle 15ᵒ*
`0xB5, 0x62, 0x06, 0x01, 0x08, 0x00, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x26`	*Transmittal of NMEA GxGST sentences through USB (Global Positioning System Fix Data)*
`0xB5, 0x62, 0x06, 0x01, 0x08, 0x00, 0xF0, 0x01, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x01, 0x2D`	*Transmittal of NMEA GPGLL sentences through USB (Geographic position, Latitude and Longitude)*
`0xB5, 0x62, 0x06, 0x01, 0x08, 0x00, 0xF0, 0x03, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x03, 0x3B`	*Transmittal of NMEA GPGSV sentences through USB (Satellites in view)*
`0xB5 ,0x62 ,0x06 ,0x17 ,0x14 ,0x00, 0x00 ,0x41 ,0x00 ,0x0A ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00, 0x00 ,0x01 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x7D ,0xDF`	*NMEA Output high accuracy*

Table 1: GPS Configuration

3.2 Receiving and parsing NMEA data

The employed board transmits GPS receiver data within the NMEA specification [4]. NMEA is a combined electrical and data specification for communication between marine electronics such as echo sounder, sonars, anemometer, gyrocompass, autopilot, GPS receivers and many other types of instruments. It has been defined by, and is controlled by, the National Marine Electronics Association. The purpose of NMEA is to give equipment users the ability to mix and match hardware and software. It is widely used among GPS / GNSS receivers as their communication standard. NMEA-formatted GPS data also makes life easier for software developers to write software for a wide variety of GPS receivers instead of having to write a custom interface for each GPS receiver.

The NMEA message output has the following *sentence structure*:

- $aaccc,c–c*hh<CR><LF>

The detail of the sentence structure is explained in Table 2. Checksum field is the 8-bit exclusive OR (no start or stop bits) of all characters in the sentence. Checksum consists of 2 characters and is represented as a hex number

character	HEX	Description
"$"	24	*Start of sentence.*
aaccc		*Addressfield. "aa" isthe talkeridentifier. "ccc" identifiesthe sentencetype.*
","	2C	*Field delimiter.*
C–c		*Data sentence block.*
"*"	2A	*Checksum delimiter.*
Hh		*Checksum field.*
<CR><LF>	0D0A	*Ending of sentence. (carriage return, line feed)*

Table 2: *NMEA Sentence structure*

We are especially interested on GNGGA, GNGLL, GNGSV, GNGSV, GNGST sentences which provide the receiver's solution, the respective accuracy, as well as several measures regarding the solution's quality such as PDOP, num of satellites in view etc.

Field	Name	Description
hhmmss.ss	UTC Time	*UTC of position in hhmmss.sss format, (000000.000 ~ 235959.999)*
llll.lll	Latitude	*Latitude in ddmm.mmmm format. Leading zeros are inserted.*
A	N/S Indicator	*'N' = North, 'S' = South*
yyyyy.yyy	Longitude	*Longitude in dddmm.mmmm format. Leading zeros are inserted.*
A	E/W Indicator	*'E' = East, 'W' = West*
X	GPS quality indicator	*GPS quality indicator* *0: position fix unavailable* *1: valid position fix, SPS mode* *2: valid position fix, differential GPS mode* *3: valid position fix, Precise Positioning Service mode* *4: valid position fix, RTK fix GPS mode* *5: valid position fix, RTK float GPS mode*
Uu	Satellites Used	*Number of satellites in use, (00 ~ 24)*
v.v	HDOP	*Horizontal dilution of precision, (00.0 ~ 99.9)*
w.w	Altitude	*Mean sea level altitude (-9999.9 ~ 17999.9) in meter*
x.x	Geoidal Separation	*In meter*
zzzz	DGPS Station ID	*Differential reference station ID, 0000 ~ 1023* *NULL when DGPS not used*
Hh	Checksum	

Table 3: *GNGGA Sentence structure*

For example, GNGGA Sentence (Global Positioning System Fix Data) provides Time, position and fix related data for a GPS receiver. The format of each sentence is as follows:

- $--GGA,hhmmss.ss,llll.lll,a,yyyyy.yyy,a,x,uu,v.v,w.w,M,x.x,M,,zzzz*hh<CR><LF>

And the respective details regarding each part of the sentence are given in the following Table 3

Therefore, following these specifications, we developed a component that parses the respective sentences and transforms them into meaningful data displayed in the respective user interface. The component is written in Microsoft C# using .NET framework 4.0 and is currently part of the project *GeonoesisGPS* class library Visual Studio project, which provides tools for managing GPS connections to and from a PC. For a complete description of the library, refer to [D14], which is part of WP6. In the following Table 4 the contents of specific NMEA sentences supported by *GeonoesisGPS* component are given:

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

Table 4: Overview of Ublox's F9P NMEA sentences processed by GeonoesisGPS

The communication with the receiver (data receiving) is performed via the USB interface (virtual com port) and the *GeonoesisGPS* class library in a timely manner.

All communications between our software and the Ublox board is performed via a *GPSHandler* object which can be attached to a parent user interface object (e.g., a windows form). The component provides methods for connecting to the COM port, configuring the device according to information of section 3.1, transfers events that are invoked from the GPS board such as COM port opening or timeout, parses NMEA data and fires GPS Fix events when new data are available.

3.3 Sending RTCM corrections:

RTCM corrections are essential for our system to achieve good position estimates from the UBlox board. RTCM corrections are gathered from NTRIP Casters (CYPOS, ATLAS, URANUS etc.) and transmitted to the receiver using the USB communication port.

Networked Transport of RTCM via Internet Protocol (NTRIP) is an application-level protocol that supports streaming Global Navigation Satellite System (GNSS) data over the Internet. NTRIP is a generic, stateless protocol based on the Hypertext Transfer Protocol HTTP/1.1. The HTTP objects are extended to GNSS data streams.

NTRIP is designed to disseminate differential correction data or other kinds of GNSS streaming data to stationary or mobile users over the Internet, allowing simultaneous PC, Laptop, PDA, or receiver connections to a broadcasting host. NTRIP supports wireless Internet access through Mobile IP Networks like GSM, GPRS, EDGE, or UMTS.

NTRIP is meant to be an open non-proprietary protocol. Major characteristics of NTRIP's dissemination technique are the following

- It is based on the popular HTTP streaming standard; it is comparatively easy to implement when limited client and server platform resources are available
- Its application is not limited to one particular plain or coded stream content; it has the ability to distribute any kind of GNSS data
- It has the potential to support mass usage; it can disseminate hundreds of streams simultaneously for up to a thousand users when applying modified Internet Radio broadcasting software
- Regarding security needs, stream providers and users are not necessarily in direct contact, and streams are usually not blocked by firewalls or proxy servers protecting Local Area Networks
- It enables streaming over any mobile IP network because it uses TCP/IP

The NTRIP Caster 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). The NTRIP Caster accepts request-messages on a single port from either the NTRIP Server or the NTRIP Client. Depending on these messages, the NTRIP Caster decides whether there is streaming data to receive or to send.

The communication with the NTRIP Caster is performed using the internet and a *Berkeley socket* which is used to establish and maintain communication between them. Contrary to all existing suggestions, RTCM data gathered by NTRIP provider are transmitted with our system to the receiver via the USB interface.

The respective software component is integrated into our *GeonoesisGPS* C# project and performs the following actions

- *Constructor*: Creates the NTRIP client given the IP and the port of the caster, the username and password to connect to the caster, and also a *GPSHandler* object
- *Connect*: Connects to the NTRIP caster via a network endpoint
- *CreateRequest*: Creates a request to the NTRIP caster
- *GetSourceTable*: Retrieves a Sourcetable data containing info about the supported by the caster mount points etc.
- *StartNTRIP*: Starts NTRIP communication between our software and the NTRIP caster, given the *MountPoint* on which information is requested
- *StopNTRIP*: Stops NTRIP communication between our software and the caster.

We have tested our approach and found out that the time required to get a fix from the board is in the range of 1 min, while this fix is maintained under good circumstances (atmospheric conditions, baseline length, GNSS antenna, multipath conditions, satellite visibility and Geometry according to [5].

3.4 Logging raw GPS Data

We have also implemented in our *GeonoesisGPS* class library a method for continuous logging of raw GNSS data provided by the receiver. This kind of data is used for post processing of GNSS data so as to provide PPK solutions with open-source software. We have extensively tested this approach and present our results in the next sections.

3.5 Communicating with the INS

Regarding the communication with the INS, we have initially been based on the XSens MT Manager Software [5]. MT Manager is compatible with all Xsens Motion Trackers. The MT Manager uses the XsensDeviceApi64.DLL or XsensDeviceApi32.DLL with the dynamic library interface. This is the same API that is provided for software development in the MT SDK. The MT Manager software is an easy-to-use software with familiar Windows user interface, which allows to (Figure 7):

- view 3D orientation in real-time
- view inertial and magnetic sensor data in real time
- view latitude, longitude, altitude plots in real time (depends on Motion Tracker used)
- monitor and compose message to and from the device via a message terminal
- export log files to other formats like ASCII and KMZ
- change and view various device settings and properties
- run a self-test to check the mechanical functions of the inertial sensors and magnetometer

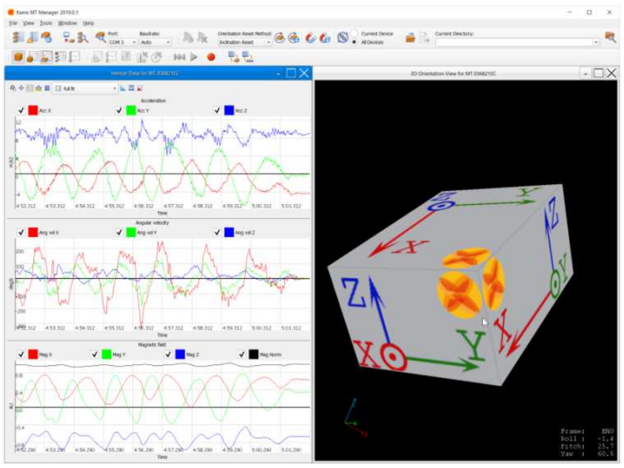


Figure 7: *MTManager screenshot*

The MT Manager is therefore an easy way to get to know and to demonstrate the capabilities of the respective motion tracker.

Specifically, we have initially used this product in order to explore the capabilities of the INS and configure it according to our needs. Since we are based mainly on GPS-provided samples for the heading calculation as described In section 5, we have configured the device based on the General Magnitude profile and saved the respective configuration file. This file can be subsequently used in order to configure the device at any time, via the provided SDK. The respective configuration can be found in Figure 8.

```
; XSA file created by Xsens (tm)
; Copyright Xsens Technologies(tm) 2006-2015
Config.AlignmentRotationLocal=N:22:FAFFEC11013F8000000000000000000000000000000044
Config.AlignmentRotationSensor=N:22:FAFFEC11003F8000000000000000000000000000000045
Config.Baudrate=N:6:FAFF180102E6
Config.FilterProfile=N:7:FAFF6402000D8E
Config.GnssPlatform=N:7:FAFF7602000089
Config.LatLonAlt=N:29:FAFF6E18404A1EC86E8C8564401B5A557F4CD161000000000000000021
Config.LocationId=N:7:FAFF840200007B
Config.OptionFlags=N:13:FAFF48080000000FFFFFFFFB5
Config.OutputConfiguration=N:21:FAFFC0101010FFFF2033000450430004502300004AE
Config.SyncConfiguration=N:29:FAFF2C180902010000000000000003E80E0801000000000001F400
00BA
Info.FirmwareVersion=S:1.4.0 build 1662 rev 84103
Info.ProductCode=S:MTi-7-8A7G6
```

Figure 8: *MTi-7 configuration*

After exploring the device and successfully configured it, we have implemented a software component that communicates with INS MTI-7, using the provided SDK (XsensDeviceApi64.DLL). Given that data gathered from the motion tracked do not follow the NMEA specification, a full .NET API is provided by the SDK enumerated using the respective measured values from the device, position, and rotation (roll, pitch, yaw). The software that we have developed performs the following actions:

- Scans com ports to find a connected device
- Establishes communication
- Configures the device
- Puts the device into measurement mode
- Handles data availability events which contain Lat Lon vectors with position error estimation, timestamps, and Orientation data (roll, pitch, yaw), and save all available data to our log files.

All such interfaces have been implemented directly into our *MobiloGrabber* project, part of D14.

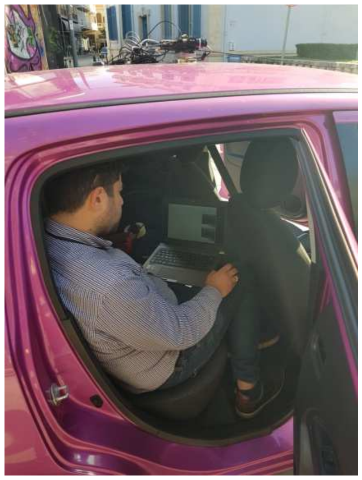
4 Test Open-Source Software Libraries for post processing.

We have employed RTK Lib software [3] for testing post processing techniques. RTKLIB is an open-source program package for standard and precise positioning with GNSS (global navigation satellite system). RTK Lib consists of a portable program library and several APs (application programs) utilizing the library. The features of RTK Lib are:

- **Supports standard and precise positioning algorithms** with GPS, GLONASS, GELILEO, QZSS, BEIDOU and SBAS
- **Supports various positioning modes with GNSS for both real-time and post-processing:** Single, DGPS/DGNSS, Kinematic, Static, Moving-Baseline, Fixed, PPP-Kinematic, PPP-Static and PPP-Fixed

- **Supports many standard formats and protocols for GNSS:** RINEX 2.10, 2.11, 2.12 OBS/NAV/GNAV/HNAV/LNAV/QNAV, RINEX 3.00, 3.01, 3.02 OBS/NAV, RINEX 3.02 CLK, RTCM ver.2.3, RTCM ver.3.1 (with amendment 1-5), ver.3.2, BINEX, NTRIP 1.0, RTCA/DO-229C, NMEA 0183, SP3-c, ANTEX 1.4, IONEX 1.0, NGS PCV and EMS 2.0
- **Supports several GNSS receivers' proprietary messages:** NovAtel: OEM4/V/6, OEM3, OEMStar, Superstar II, Hemisphere: Eclipse, Crescent, u-blox: LEA-4T/5T/6T, SkyTraq: S1315F, JAVAD: GRIL/GREIS, Furuno: GW-10 II/III and NVS NV08C BINR (refer the Manual for details)
- **Supports external communication via:** Serial, TCP/IP, NTRIP, local log file (record and playback) and FTP/HTTP (automatic download)
- **Provides many library functions and APIs for GNSS data processing:** Satellite and navigation system functions, matrix and vector functions, time and string functions, coordinate transformations, input and output functions, debug trace functions, platform dependent functions, positioning models, atmosphere models, antenna models, earth tides models, geoid models, datum transformation, RINEX functions, ephemeris and clock functions, precise ephemeris and clock functions, receiver raw data functions, RTCM functions, solution functions, Google Earth KML converter, SBAS functions, options functions, stream data input and output functions, integer ambiguity resolution, standard positioning, precise positioning, post-processing positioning, stream server functions, RTK server functions, downloader functions

Among others RTK Lib includes GUI and CUI APs for Real time positioning (RTKNAVI), Post-processing analysis (RTKPOST), Rinex converter (RTKCONV), Plot positions and observation data (RTKPLOT). All of the executable binary APs for Windows are included in the package as well as whole source programs of the library and the APIs. RTK Lib is currently the only available open-source software for processing GPS/GNSS data.


(a) Experimental setup

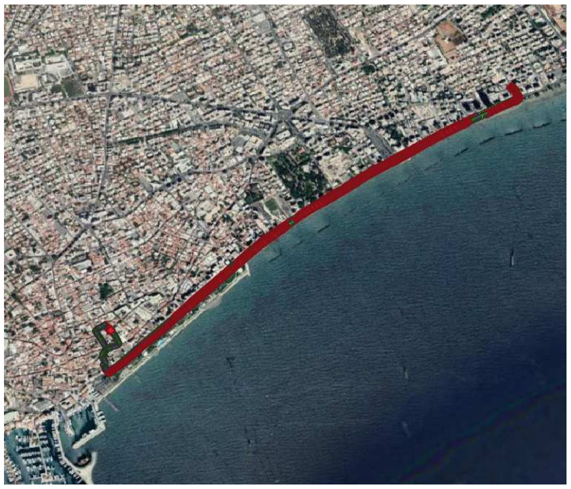

(b) Test site

Figure 9: Setup and test site for experimental data gathering regarding GPS solutions

4.1 Data Collection

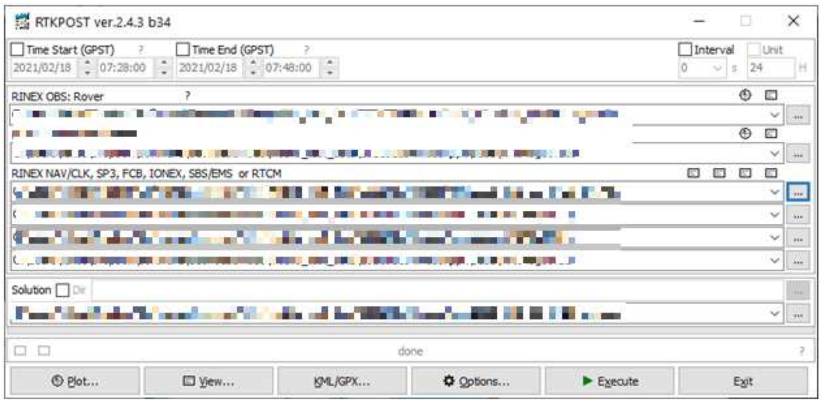
We have performed several experimental data gathering so as to compare the performance of solutions provided by RTK Lib vs the solution provided by our Ublox board [5]. Specifically, for the collection process, we mounted our preliminary system onto a car and collected data using our software library. During the data collection process, we were connected to the ATLAS NTRIP caster providing RTCM message to the F9P board so as to achieve RTK solutions. The solutions provided along with the board raw data were stored into the data collection device (laptop). The test site was chosen to be in the sea front of Limassol in order to achieve open air conditions (Figure 9). The solutions provided by F9P are summarized in Table 5. We have successfully reproduced the same results under similar conditions (rather low constructions and vegetation) .

Fix status	Number of observations	Percentage (%)
Fixed	1562	64.07%
Float	731	29.98%
DGPS	145	5.95%
No Solution	0	0.00%
Totals	**2439**	**100%**

Table 5: Fix status and number of observations in the test site for solutions provided by F9P RTK Solutions

4.2 Data Processing with RTK Lib

In order to post process raw GNSS data, we have also obtained from the ATLAS network observations from a virtual CORS station nearby our test site. We subsequently used RTKCONV to convert u-blox data into RINEX format, and RTKPOST to process our data with PPK kinematic positioning mode and combined method (forward and backward). An example screenshot of the respective RTKPOST software can be found in Figure 10.


RTKPOST ver.2.4.3 b34

Time Start (GPST) 2021/02/18 07:28:00 — Time End (GPST) 2021/02/18 07:48:00 — Interval 0 s — Unit 24 H

RINEX OBS: Rover [redacted]

RINEX NAV/CLK, SP3, FCB, IONEX, SBS/EMS or RTCM [redacted]

Solution [redacted]

done

Figure 10: RTKPOST example screenshot

Due to several limitations and bugs of the open source RTK Lib we concluded to use different software versions for converting UBX data to RINEX [6] with RTKCONV and then proceed with their kinematic solution with RTKPOST (and the respective CUIs, e.g., CONVBIN and RNX2RTKP). Table 6 summarizes the above information.

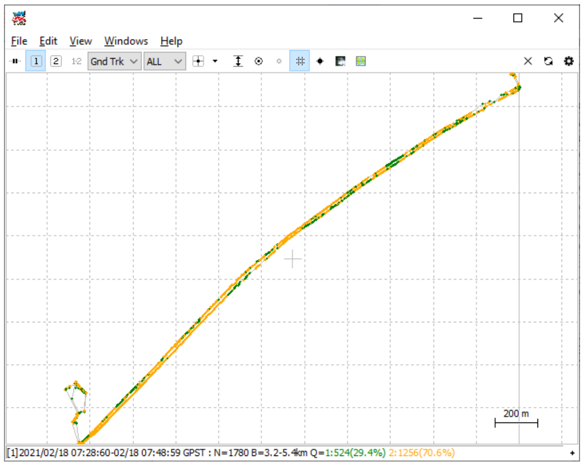
GUI	CUI	RTK Lib Version	Source	Usage
RTKCONV.exe	CONVBIN.exe	b33f	[7]	Convert to RINEX format
RTKPOST.exe	RNX2RTKP.exe	2.4.3 b34	[3]	Process baselines

Table 6: RTK Lib executables used

The solutions provided by RTK Lib regarding the data provided in section 4.1, are summarized in Table 7. Solutions provided by RTKPOST and RNT2RTP can be visualized with RTKPLOT (Figure 11).

Fix status	Number of observations	Percentage (%)
Fixed	523	21.45%
Float	1296	53.16%
DGPS	0	0.00%
No Solution	619	25.39%
Totals	**2439**	**100%**

Table 7: Fix status and number of observations in the test site for solutions provided by post processing with RTK Lib



[1]2021/02/18 07:28:60-02/18 07:48:59 GPST : N=1780 B=3.2-5.4km Q=1:524(29.4%) 2:1256(70.6%)

Figure 11: Visualization of solutions provided by RTK Lib with RTKPLOT

4.3 Data Processing with commercial software

We have further employed Novatel's GrafNav software [8] which is a powerful commercial post-processing software. GrafNav post-processing software is a powerful, highly configurable processing engine that allows for the best possible static or kinematic GNSS accuracy using all available GNSS data. Support of data formats from most single and multi-frequency commercial receivers means GrafNav will likely work with your existing hardware. A full suite of data and solution visualization and diagnostic tools is available for quality assurance.

We have therefore used the same ATLAS network observations from a virtual CORS station nearby our test site as described in 4.2 as well as the RINEX data regarding out receiver, obtained by RTKCONV. The results of the respective solution can be found on Table 8.

Fix status	Number of observations	Percentage (%)
Fixed	236	9.68%
Float	464	19.03%
DGPS	1099	45.08%
No Solution	639	26.21%
Totals	**2439**	**100%**

Table 8: Fix status and number of observations in the test site for solutions provided by post processing with GrafNav

4.4 Solution's Comparison

At a first look it seems that RTK solutions provided by F9P outperform both post processing solutions by RTK Lib and GrafNav in terms of percentage of fixed solutions (64%>21.45%>9.68%). Moreover, it seems that RTK Lib outperforms GrafNav in the same terms; however, a closer look at the results reveals several problems in the solutions provided by RTK Lib. Figure 12 displays a screenshot of the solutions provided by F9P (magenta) and RTK Lib (blue); diamonds display fixed solutions, while dots display float solutions (for both methods). Figure 12 makes clear that RTK Lib fixed solutions differ significantly from their previous and next solutions, and many of them lie on obviously incorrect positions. Moreover, GrafNav float solutions lie are rather erroneous, while F9P provided positions display a fixed and / or float logical sequence solution.

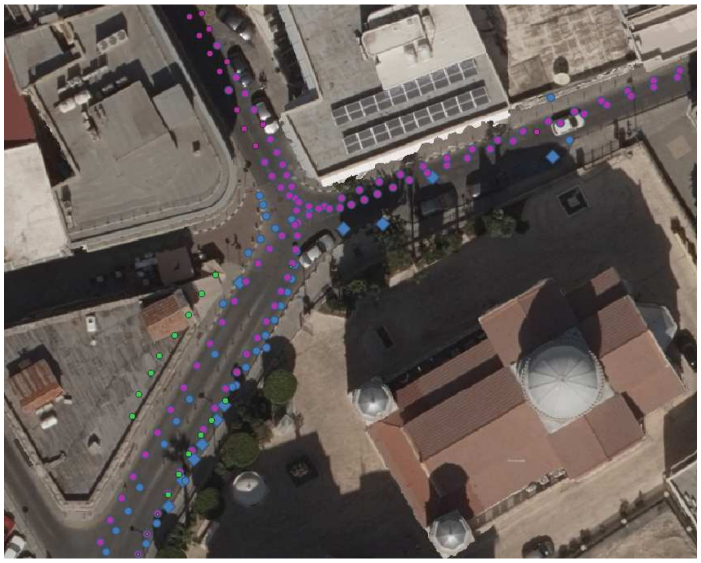


Figure 12: Visualization of solutions provided by RTK Lib (blue), F9P (magenta) and GrafNav (green)

Since float solutions do not provide a ground truth for testing purposes, we compared only fixed solutions of all three alternatives. Table 9 summarizes the results of our comparison where we calculated the mean distance between corresponding solutions (i.e., on the same timestamp). It is clear that solutions provided by F9P and GrafNav are compatible between them, e.g., difference between provided solutions are in the order of 1 – 2 cm. On the other hand, solutions provided by RTK Lib differ significantly from the other two alternatives. All these observations direct us to conclude that RTK Lib fix positions are not to be trusted and may differ up to several centimeters from the actual position.

	# observations	Mean distance XY	Mean distance Z
F9P / GrafNav	209	0.01	0.02
RTK Lib / GrafNav	56	0.04	0.26
F9P / RTK Lib	417	0.18	0.37-

Table 9: Difference (m) between fixed solutions of all alternative solution providers

4.5 Additional Experiments

In order to test all alternative solutions, we deigned another experiment with the aim of a standard geodetic GPS / GNSS Leica Viva. Specifically, our aim was to achieve simultaneous solutions by all methods. Therefore, we mounted on top of a vehicle both F9P antenna and Leica Viva GPS / GNSS receiver (Figure 13); the latter was configured to log data so as to post process them. Then we visited our test site and performed experimental data gathering collecting both RTK and raw GPS / GNSS data.

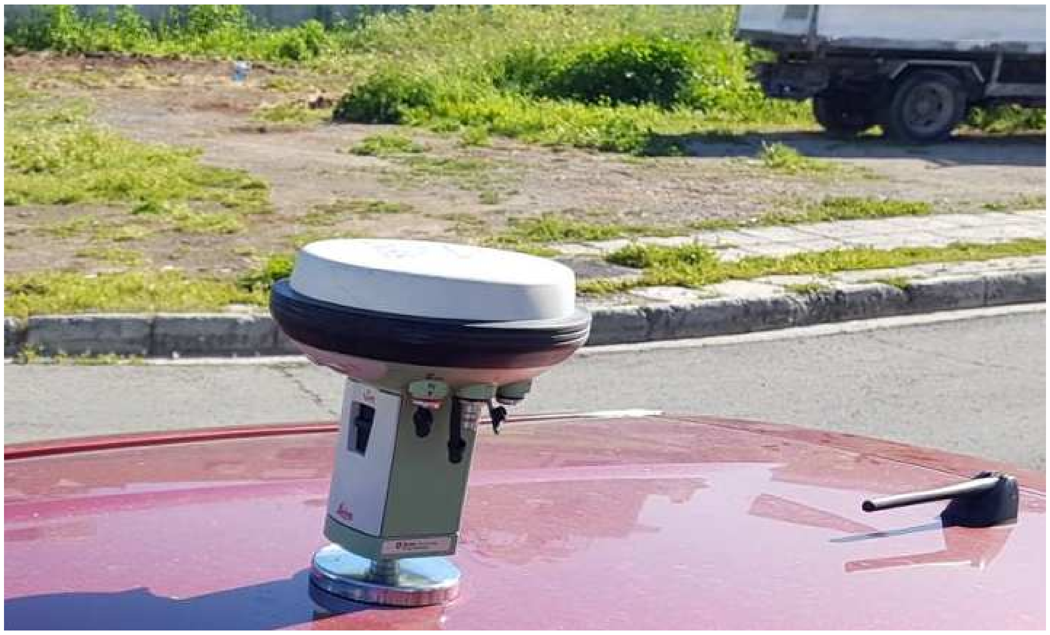


Figure 13: Experimental setup for GPS data comparison

Given that both receivers are mounted in stable positions in top of the vehicle, the vector of their difference should maintain a stable value throughout the experimental data gathering, regardless of the vehicle's heading. Under these conditions it is expected to obtain stable XY and Z distances between the two receivers, and the respective standard deviations should me minimum. Table 1 summarizes the results obtained by all alternatives. RINEX data provided by Viva receiver were processed as kinematic solutions. Specifically, Table 1 reveals that fixed solutions provided by F9P are true fixes since the standard deviation of the XY distance does not exceed 0.01 m, while vertically the respective value is 0.04. On the other hand, distances between the two antennas, provided by post processing solutions of RTK Lib with the data obtained by F9P board (and the same ground truth data), although marked as fixed, show very high standard deviation

Ground Truth data		F9P	RTK Lib
Leica Viva (fix solutions)	Fixed Observations	98 (100%)	8 (10%)
	σ (distance XY)	0.01	1.46
	σ (distance Z)	0.04	1.41

Table 10: Standard deviation of measured distance between VIVA receiver and F9P's antenna for fixed solutions as well as solutions provided by RTK Lib.

We have successfully reproduced the same results under similar conditions (rather low constructions and vegetation) and concluded that RTK Lib does not improve the solutions provided by F9P, which are directly comparable with the solutions provided by standard expensive geodetic GPS / GNSS receivers.

5 Improving Heading Calculations

In order to improve heading calculations, we implemented an approach based on the fixed RTK positions provided by the GNSS component of our system. Our method is based on a two-step approach where the car's movement heading vector is initially calculated by consecutive GNSS solutions (Axis *X* in Figure 14), and then a correction angle $\delta_{heading}$ is added to this estimation in order to determine the final system's heading (axis *x* in Figure 14)

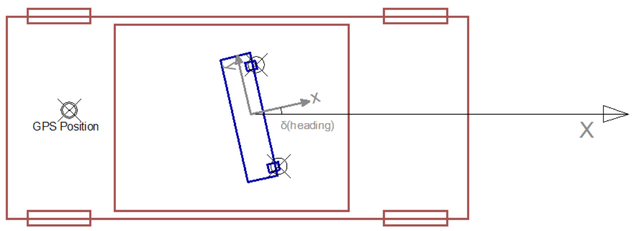


Figure 14: Calculating MOBILO's system heading

5.1 Movement heading vector

Regarding the calculation of the car's movement heading vector, a straightforward approach implies using the previous and current GNSS position to calculate the heading vector of movement. While this can be considered as an acceptable approach on a real-time environment, in our case we may use the knowledge of the previous and subsequent position to gather a more accurate estimation.

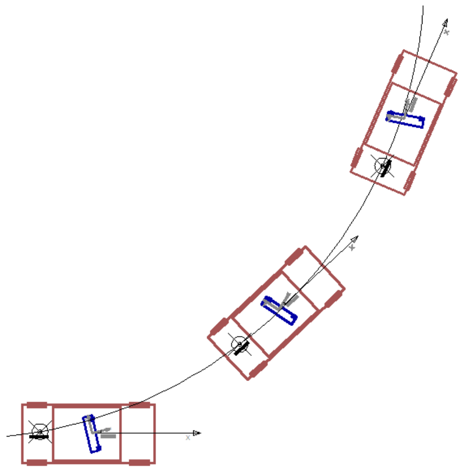


Figure 15: Arc calculation and tangent heading

Specifically, given that MOBILO system is typically mounted on top of a moving vehicle we expect that its movement will follow a rather canonical pattern, i.e., moving on straight lines and circular rings. Therefore, given the previous, the current and the subsequent vehicle's position, we may use an *arc* lying onto these 3 points and then determine the *arc's tangent* on the current sampled position (Figure 15).

Moreover, given a higher number of points, we may calculate the *least squares arc* that lies between all given points. For this calculation we apply the general circle equation

$$(x-x_0)^2+(y-y_0)^2=R^2 \Longrightarrow\ 2xx_0+2yy_0+(R^2-x_0^2-y_0^2)=x^2+y^2$$

By substituting $c=(R^2-x_0^2-y_0^2)$, we obtain $2xx_0+2yy_0+c=x^2+y^2$, where *x*$_0$, *y*$_0$ and *c* are linearly depended by the observations and can be determined by general least squares

$$M\times z=y\text{, where}$$

$$M=\begin{bmatrix}2x_1 & 2y_1 & 1\\ .. & .. & ,..\\ 2x_n & 2y_n & 1\end{bmatrix}, z=\begin{bmatrix}x_0\\ y_0\\ c\end{bmatrix}, y=\begin{bmatrix}x_1^2+y_1^2\\ \dots\\ x_n^2+y_n^2\end{bmatrix}.$$

Then we use Gauss elimination to solve the problem for *z*

$$z=(M^T\times M)^{-1}\times M^T\times y,$$

and finally determine *x*$_0$, *y*$_0$, and *R*

Investing on the above observations we have ended up with a heading calculation algorithm that can be summarized in the following Figure 16.

```
double GetAzimuth(Coordinate coordinate, double arcMinimumLength = 0) {
           // search candidate segments for correct point location
01.        Coordinate candidate = GetClosestCoord(trajectory, coordinate);
           // set initial arc locations by the index of the queries point
           // queried point is placed on the center (i2)
02.        i1,i2,i3 = trajectory.IndexOf(candidate);
           // calculate arc based on the following conditions
           // a) we have reached the end (or the start) of the trajectory, OR
           // b) arcminimumlength is not provided at all, OR
           // c) we have exceeded arcminimumlength /2 on both sides of the
           //    calculated arc
03.        do {
               //set arc start to the previous location
04.            if (i1 > 0) i1 -= 1;
               //set arc end to the next location
05.            if (i3 < trajectory.Count - 1) i3 += 1;
06.            if ((i3 - i1) >= 2) arc = new Arc(trajectory,i1,i2,i3);
07.        } while (!((i3-i1) >=2 && (arcMinimumLength == 0 || (
                      arc.Length > arcMinimumLength &&
                      arc.p1.Distance(arc.p2) > arcMinimumLength / 2 &&
                      arc.p2.Distance(arc.p3) > arcMinimumLength / 2)) ||
                       (i1 == 0 && i3 == base.Count - 1)));
           // determine arc’s direction (clockwise or counterclockwise)
08.        ccw = arc.CCW ? -1 : 1;
            // calculate error of used points
09.        error = CalculateError(i1,i3,trajectory,arc);
            // the resulted azimuth is perpendicular to the center -
           // last point vector
10.        angle = new Angle(arc.Center, coordinate) + ccw * Math.PI / 2;
            // return result with calculated error
11.        return (angle, error);}
```

Figure 16: Calculating vehicle’s heading

The algorithm calculates Azimuth of given point on an interpolated arc produced by the closest trajectory points. Moreover, if *arcMinimumLength* parameter is set, the algorithm may use points that are distant from the given coordinate so as to achieve the minimum required arc length.

The algorithm initializes by determining the closest trajectory point to the point requested by the algorithm (line 1). It then iteratively moves to the next and the previous trajectory point, calculating the least squared fitted arc to the respective trajectory points lying between indexes *i*1 and *i*3, until the *arcMinimumLength* parameter, representing the minimum length of the interpolated arc, becomes less than the currently calculated arc length (lines 3 -7). Moreover, this parameter should be satisfied at its half, regarding backward and forward distances from the requested point. Finally, the algorithm determines whether the calculated arc is *clockwise* or *counterclockwise* (line 8) and returns the respective tangent to the requested point (lines 10-11), which represents the vehicle’s heading.

5.2 Heading calibration: Calculating the correction of moving heading vector

In order to determine the correction angle $\delta_{heading}$ that is added to the vehicle’s movement heading so as to estimate the MOBILO system’s actual heading, we assume the existence of an actual linear object that is visible from multiple frames on our imaging subsystem (Figure 17). This is a rather valid assumption since this kind of objects are usually visible in road images obtained by a moving vehicle, e.g., road fringes, linear delineations etc.

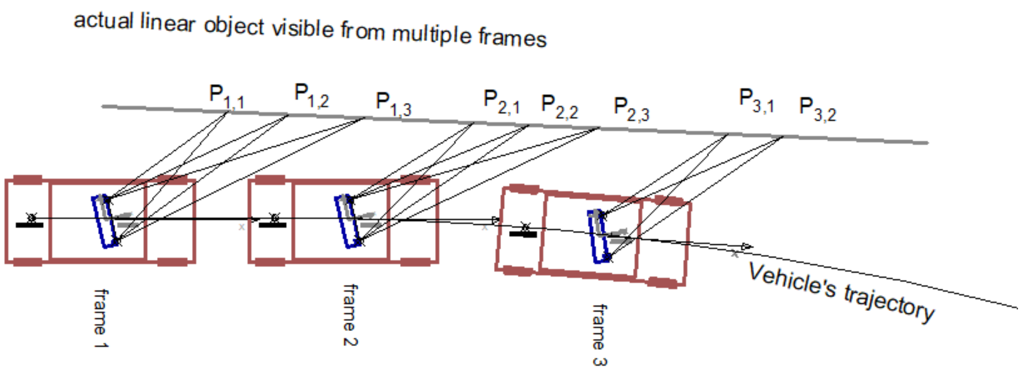


Figure 17: Points' positions that lie on a straight line, after adding $\delta_{heading}$ to the MOBILO's system heading

Figure 18 illustrates our scenario. According to our initial assumption $\delta_{heading}$ is set to zero, i.e., we assume that MOBILO system is set parallel to the vehicle's movement, and we locate on multiple frames several points ($P_{1,1}$, $P_{1,2}$, $P_{1,3}$, ... $P_{3,2}$) that lie on a straight line. Obviously, while points that lie on a linear object gathered from the same frame should also form a straight line (e.g., points $P_{1,1}$, $P_{1,2}$, $P_{1,3}$), this is not true for points gathered from multiple frames (e.g., points $P_{1,1}$, $P_{1,2}$, $P_{1,3}$ and $P_{2,1}$, $P_{2,2}$, $P_{2,3}$ on Figure 18) since $\delta_{heading}$ is set to zero.

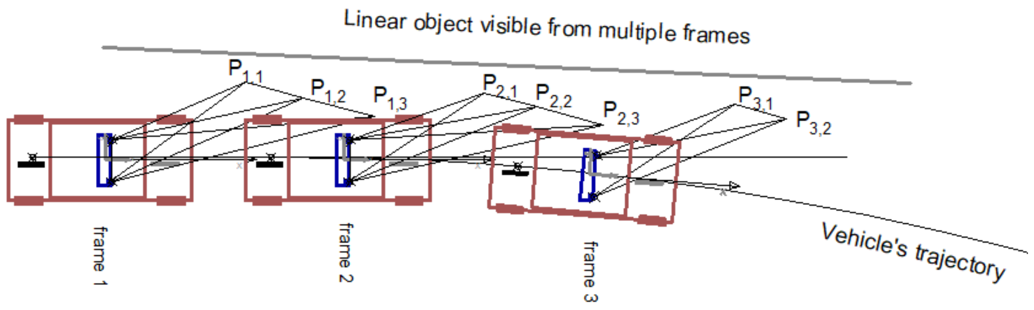


Figure 18: Determining points' positions that lie on a straight line, before adding $\delta_{heading}$ to the MOBILO's system heading

We have therefore developed an approach on the calculation of $\delta_{heading}$ implemented by the recursive algorithm illustrated in Figure 19. The algorithm divides an initial given relatively large *angle* and determines *n* possible angles spanning between [–angle, +angle] (lines 01-05). Each one of the *n* possible angles, is applied into the calculation of points' ($P_{i,i}$) world coordinates as $\delta_{heading}$, and then, the error of the world points from the best-fitted straight line among $P_{i,i}$s is calculated (lines 06 – 22). Having determined the *k*-th *angle* that provides the best-fitted line among $P_{i,i}$, it is set as the current $\delta_{heading}$ and the algorithm recursively call itself with an *angle* that is calculated as *angle* / *n*, i.e., it focuses on the interval that has its center the *k*-th angle and spans between *k*-1 and *k*+1 angles (lines 22 – 28).

```
void CalibrateHeading(List<Points> PointsOnLine, Angle range, double toler) {
        // initialize variables
        LeftMostPhi = ProjectHeading - range; minerr = double.Maxvalue;
        // divide value of range into 10 spaces
        for (int i = 0; i <= 10; i++)
        {
           // set current value of candidate heading
            candidateHeading = LeftMostHeading + range * i / 5;
           // initialize list of world coordinates based on candidateHeading
            LeastSquareCoords = new List<Coordinate>();
            // for each point in PointsOnLine apply candidateHeading in respective
            // frame's exterior and calculate world coordinates
            foreach (point in PointsOnLine)
            {
                Frame = point.Frame;
                Frame.Exterior.phi += candidateHeading;
                foreach (point in Frame)
                {
                    point.CalculateWorld ();
                    // add world coordinates into current list
                    LeastSquareCoords.Add(point);
                }
            }
            // calculate linear error by deming regression
            err = CalculateLineErr(lst);
            // if error is less than the current minimum replace its value
            if (minerr > err)
            {
                minHeading = candidateHeading;
                minErr = err;
            }
        }
        // check whether difference between ProjectHeading and minHeading is
        // less than the tolerance, i.e., has been only marginally changed or
        // range /5 is less than the tolerance so as to stop
        if (Abs(ProjectHeading - minHeading) > toler || range / 5 > toler)
        {
            // otherwise recursively call itself with 5 times smaller range
            ProjectHeading = minext;
            CalibrateHeading (PointsOnLine, range / 5, toler);
        }
}
```

Figure 19: Algorithm 1 for determining $\delta_{heading}$

This error of the straight line that is best fitted among $P_{i,j}$ is determined via Deming regression [9], which is an errors-in-variables model that tries to find the line of best fit for a two-dimensional dataset. It differs from the simple linear regression in that it accounts for errors in observations on both the x- and the y- axis (Figure 20).

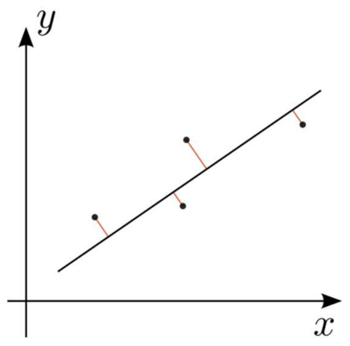


Figure 20: Deming regression

We have extensively tested this approach and determined that the error determined by the Deming regression is in the order of 0.1 m, that is, we finally obtain a very good approximation of the straight line. Figure 21 illustrates the observed positions of several points aimed from several frames of MOBILO system, before (a) and after (b) applying the algorithm of Figure 19 that is finally used for determining $\delta_{heading}$ (heading calibration). Here it must be noted that the *heading calibration* procedure presented in the previous paragraphs, must be executed every time the MOBILO system is mounted on top of the moving vehicle, and is valid for all data collected by the system, unless it is somehow moved from it (intentionally or unintentionally).

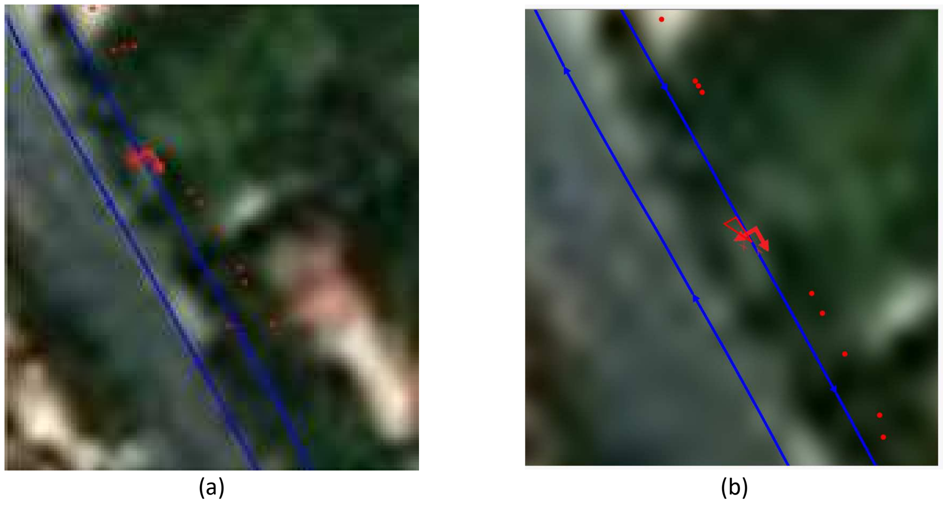


Figure 21: Observed positions of points lying on straight before (a) and after (b) heading calibration.

6 Future Work

Following several suggestions, we plan to test an additional solution exploiting 3 GPS receivers. We believe that the respective cost increase does not excel our original planned cost per system. We will investigate into a solution employing three RTK devices which will provide not only the position of the system under development, but also, the respective rotation. While this solution will increase the development cost by at least 1000 €, it will provide very good results under fixed conditions. This means that in rural environment we may get rid of the solutions provided by INS and use the much more accurate fixed solutions provided by the three receivers in order to compute the camera's external orientation. We cannot completely dispose the INS from our system, since in urban – forest environments it is not expected to get fixed solutions by the board, therefore the employment of INS calculated positions and rotations is the only feasible choice.

7 Conclusion

Aim of this report is to demonstrate the positioning methods integrated into of MOBILO's system. According to the submitted proposal, *MOBILO proposes a low-cost mobile mapping system which consists of a GPS / GNSS RTK, an inertial INS / IMU system which gathers position and orientation data, as well as video cameras to collect image data*. WP4 is responsible for the positioning subsystem of our system which also includes the calculation of the exterior orientation based on INS and / or other methods presented in this report.

Regarding the specific tasks of WP4, these are closely related to the tasks of WP3, focusing on the continuous calculation of the accurate exterior orientation of the total system, which will be used to provide the respective exterior orientation of the imaging subsystem by adding the misalignment between the cameras and the mapping reference system calculated in WP3.

According to the Annex I of the contract, three tasks were to be performed during this WP execution, namely, which are addressed as follows:

- **<u>A4.1. Decision of the GPS and INS specifications:</u>** Several recently available low-cost high precision GNSS boards in the market with their cost in the order of hundreds of euro, making it therefore easy to integrate them into our system. We invested into U-blox's high precision F9P board with accuracy of 0.01 m+ 1 ppm CEP horizontally and 0.01m +1 ppm CEP vertically. We also obtained INS XSENS MTI-7-DK board which measures data rotation data with an RMS of Roll/pitch (static – dynamic) 0.5° and Yaw (dynamic) 1.5°. We physically integrated these boards into a *project box* with an appropriate antenna connected to both.
- **<u>A4.2: Setup and trials for experimental data gathering:</u>** We first explored the interface of all employed boards using the end-user software provided by their vendors. We subsequently developed several software components that communicate with the device, configure it, send to it RTCM messages provided by a NTRIP caster, read position solutions provided by the boards and store raw data. We established communication with the boards with their USB interface forming virtual COM interfaces. Communication with the GPS / GNSS is based on the NMEA specification via standard COM ports using C# and .NET framework on Windows PC, while communication with the INS is established using the respective SDK provided by the board's vendor.
- **<u>A4.3: Test existing open-source Software libraries for post processing:</u>** We setup a test site and performed experimental data gathering using several configurations: F9P board with RTK solutions, post processing of raw data collected by F9P board with open-source libraries, i.e., RTB Lib and commercial software such as Novatel's GrafNav, as well as simultaneous data gathering with standard geodetic GPS / GNSS receivers, i.e., Leica Viva, with the RTK method. The comparison between all alternative shows that RTK solutions provided by U-blox F9P board is directly comparable with the solutions provided by standard expensive GPS / GNSS, while post processing with RTK Lib does not achieve acceptable results that can be used in our system.

Along with the tasks that where predefined in the project's Annex I, a set of other tasks arouse during the project's implementation, which significantly improve the performance of our system:

- **<u>Improving heading calculation</u>**: Specifically, given that MOBILO system is typically mounted on top of a moving vehicle we expect that its movement will follow a rather canonical pattern, i.e., moving on straight lines and circular rings. Therefore, given the previous, current, and subsequent vehicle's positions, we may use a least squares *arc* lying onto these points and then determine the *arc's tangent* on the current sampled position. We further introduce an algorithm that calibrates system's heading relatively to the vehicle's heading and significantly improves heading vectors provided by the INS.

8 References

[1] UBlox, "Ublox ANN-MB Series, High precission GNSS Antennas," [Online]. Available: https://www.u-blox.com/en/product/ann-mb-series. [Accessed 21 February 2021].

[2] "U-center," U-blox, [Online]. Available: https://www.u-blox.com/en/product/u-center. [Accessed 31 3 2019].

[3] "RTK Lib," 28 2 2020. [Online]. Available: http://www.rtklib.com/.

[4] National Marine Electronics Association, "NMEA Standards," [Online]. Available: https://www.nmea.org/content/STANDARDS/NMEA_0183_Standard. [Accessed 2 2020].

[5] Ublox, "ZED-F9P," UBlox, [Online]. Available: https://www.u-blox.com/en/product/zed-f9p-module.

[6] XSens, "MT Manager," 5 10 2020. [Online]. Available: https://www.xsens.com/software-downloads. [Accessed 5 10 2020].

[7] IGS, "RINEX," 28 2 2020. [Online]. Available: https://www.igs.org/wg/rinex/. [Accessed 28 2 2020].

[8] RTKLibExplorer, "RTKLibExplorer," 28 02 2020. [Online]. Available: https://rtklibexplorer.wordpress.com/. [Accessed 28 02 2020].

[9] "GrafNav," 2020. [Online]. Available: https://novatel.com/products/waypoint-software/grafnav. [Accessed 28 02 2020].

[10] W. E. Deming, Statistical adjustment of data, NY : Wiley, 1943.