Implementation of Inertial Navigation Algorithm into Robot Operating System (ROS)

Summer work experience, duration: 5 weeks

Author: Bc. Michal Přibil

Supervisor: Ing. Michal Reinštein, Ph. D.

Abstract

Inertial navigation is one of the alternatives to specifying objects position without having virtually any information from the outer world. In this case, the object is the mobile robot from international NIFTi project. The reason to use inertial navigation is that we need to record the robot movement with high dynamics.

The result of this work is a C++ implementation of an inertial navigation system (INS) mechanization algorithm, which processes data collected by the Xsens MTi-G inertial measurement unit (IMU).

Contents

1 Introduction

The object of our work – the NIFTi robot (see http://www.nifti.eu/) – uses a number of sensors to help specify its position. Let's start with odometry sensors that measure rotation of robot's motors. Using these sensors we can determine robot's speed and rotation about its vertical axis, but there are some reasons, why using odometry isn't enough. Firstly, knowing the rotation about the vertical axis is useless without knowing what the axis direction is. Secondly there may be some cases when the robot movement doesn't correspond with motors rotation, e.g. slipping or falling.

The following two sensors are located in the IMU's package beside the inertial sensors (triaxial accelerometer and triaxial angular rate sensor). First of them, the GPS (Global Positioning System) receiver allows us to determine the robot's position, but hardly it's orientation. The second one – triaxial magnetometer – can be used to determine orientation based on Earth's magnetic field, but this method is very inaccurate and unreliable due to possible disturbances created by metal objects and sources of parasitic magnetic fields, such as motors used to drive the robot.

The reason for using inertial navigation is obvious – to keep track of robot's orientation using triaxial angular rate sensor and to record high movement dynamics using triaxial accelerometer.

The aim of this work was to implement two versions of the INS mechanization (according to [1]) for ROS (definition from [2]: *„ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is a completely open source (BSD) and free for others to use, change and commercialize upon.“*). In order to properly understand this work, there must be two additional terms explained. Definition of the first of them – **node** – can be found at the site [3]: *„A node is a process that performs computation.“*. The other – **topic** – is defined by [3] as well: *„Topics are named buses over which nodes exchange messages.“*

First of the implemented nodes – **ins –** implements only attitude mechanization and was intended to be a final product, the other node – **inso –** extends the **ins** node to complete INS mechanization (adding dual integration of acceleration to compute position) and is about to be extended further by implementing advanced non-linear state estimation methods using GPS and odometry data.

2 Software requirements

This chapter describes in its first part, how the program communicates with the rest of the system, and in the second, how the acquired and processed data are saved to files.

2.1 System interface

As it was said previously the robot we're working on is powered by the ROS middleware. The reason to use this system is that we can simply publish data, listen to them and display them. The next reason is that we can run the nodes from different machines and monitor them.

Both implemented nodes subscribe to topics listed in Table 1. The received data are processed and results are published to topics described by Table 2.

Topic name	Data contained
/mtig_node/imu/data	Inertial data measured by Xsens MTi-G IMU
/mtig_node/pos_nav	GPS data measured by Xsens MTi-G IMU
/odom	Data collected by odometry sensors

Table 1: Subscriber topics for both ins and inso nodes

Topic name	Data contained
/mechanization_output	Euler angles (degrees), quaternion and direction cosine matrix for transformation from body frame to navigation frame For **inso** node only there is also the position in navigation frame (*m*) published
/tf	standard topic used to broadcast transformations, for **ins** node only orientation, for **inso** node both orientation and position in the navigation frame are broadcasted

Table 2: Publisher topics for both ins and inso nodes

2.2 Data logging

Both nodes also output all the data (from subscribed and published topics) into files defined by ROS parameters according to Table 3.

ROS parameter name	Effect if it is defined
ins_mech	IMU data (subscribed topic */mtig_node/imu/data*) and mechanization output (published topic /*mechanization_output*) are logged to file: ***ins_mech*(*current date & time*)**
ins_gps	The GPS data (subscribed topic */mtig_node/pos_nav*) are logged to file: ***ins_gps*(*current date & time*)**
ins_odometry	The odometry data (subscribed topic */odom*) are logged to file: ***ins_odometry*(*current date & time*)**

Table 3: Setting of data log files

The data are saved line by line to the output files. Each line is constructed according to pattern displayed in Fig. 1. Each two numbers in the log file are separated by comma. During the calibration period, EUL_MECH and PN are logged as zeros.

ins node

IMU-computed orientation NOT logged

TIME_STAMP	ACC	GYR	EUL_MECH

IMU-computed orientation logged

TIME_STAMP	ACC	GYR	EUL_MECH	EUL_IMU

inso node

IMU-computed orientation NOT logged

TIME_STAMP	ACC	GYR	EUL_MECH	PN

IMU-computed orientation logged

TIME_STAMP	ACC	GYR	EUL_MECH	EUL_IMU	PN

ACC – 3 accelerations (m/s^2), GYR – 3 angular rates (rad/s), EUL_MECH – 3 Euler angles computed by mechanization (deg), EUL_IMU – 3 Euler angles computed by IMU (deg), PN – 3 positions in NED (m)

Figure 1: Format of the mechanization output file

3 Program configuration and execution

The both nodes accept arguments to control the program behavior or to show the help (see Table 4) . Paths to logging files are parsed by ROS parameters (if they are defined) when the program is executed: list of parameters including description is available in Table 4.

Short (-) and long (--) switch	Effect of the argument
-h, --help	Shows help with list of arguments and exits
-cf, --config-file	Followed by specified path to configuration file (REQUIRED)
-y, --yaw	Specify initial yaw euler angle (degrees), depending on following value: – [none] – yaw determined automatically – [specified yaw value (range = <-180,180>)] – load specified value
-af, --attitude-feedback	choose the attitude feedback method, depending on following value: – [none] – attitude feedback disabled – avg – weighted averaging using accelerations data – fil – filtering feedback
-il, --internal-logged	log the IMU-computed orientation data

Table 4: List of program arguments

3.1 Default launch file and its usage

Launch files are used in ROS to automate node launching process. The default launch file in **ins** package is displayed in Fig. 2, the only change to launch file in **inso** package is in the name of the node.

This launch file sets up the parameters for log files paths (discussed in 2.2) and executes the node with the desired arguments. In this case the arguments are: *-cf* , which is required,

followed by the path to the configuration file; *-il* – the IMU-computed orientation data are logged; and *-y* setting the initial yaw value to 0 degrees (can be changed to obtain the initial heading information from the magnetometer measurements).

Using default launch file is sufficient and both nodes start correctly and process the data published by **mtig_node**, if there are any.

```
<launch>

  <param name="ins_gps" value="/home/robot/ctu_logger/ins_gps" />
  <param name="ins_odometry"value="/home/robot/ctu_logger/ins_odometry" />
  <param name="ins_mech" value="/home/robot/ctu_logger/ins_mech" />

  <node name="insnd" pkg="ins" type="ins" args="-cf
       /home/robot/workspace/ins/mechanization_config -il -y 0.0"/>

</launch>
```

Figure 2: Default launch file for **ins** package

The default launch files are located in directory *path_to_package/launch/* with respective names (*ins.launch* or *inso.launch*).

Using launch file:

The launch file is used by typing *„roslaunch ins ins.launch“* (for inso: *„roslaunch inso inso.launch“*) into console.

3.2 Launching pre-requisites

In order for the nodes to work and publish relevant data, the basic steps must be made:

a) *roscore* command

- this command must be executed before any nodes can be launched. It starts the ROS Master, ROS Parameter Server and *rosout* logging node (for more detailed description, see the web site [3])

b) */mtig_node/imu/data*, */mtig_node/pos_nav*, */odom* topics published to

- with the a) pre-requisite satisfied the nodes would start correctly, but would be idle until there are data published to topics the nodes subscribe to. This can be accomplished either by launching **mtig_node** or by using *rosbag* utility to play the

previously recorded data in the form of *bagfiles* (more details on *rosbag* again available at [3]).

NOTE: Both pre-requisites are required for using default launch file or running the node using *rosrun* command. However, the launch file can be customized to execute *roscore* command and even launch **mtig_node** or start playing back the data using *rosbag*.

4 Implementation

This part of the document shows how the nodes were constructed. Description of the source files, the original algorithm and the node concept are discussed here.

4.1 Source code structure

The source code is divided into 4 files:

ins.cpp (inso.cpp) – contains main function of the node and SharedObjects class

definitions.h – holds constants and data type definitions (see Appendix A)

functions.cpp – implementations of almost all functions used in the project

functions.h – header used to include functions.cpp prototypes (see Appendix B)

4.2 Resources

The algorithm of INS mechanization was supplied by M. Reinštein, PhD. in the form of MATLAB script: all the required Mathworks and non-Mathworks functions were included. The MATLAB implementation was realized for post-processing of the inertial data, but the resulting C++ implementation was required to run strictly real-time.

4.3 MATLAB data types and operations emulation

For maximum simplification of implementation of MATLAB vector, matrices and operations with them, the open computer vision library (**OpenCV**) was used. Namely, for vector and matrices the **cv::Matx** template class was used. All the vectors used are columns, so they are implemented as one-column **cv::Matx**. Multiplications are realized with standard ***** operator, transposing with **t()** function, both using OpenCV.

4.4 Node concept

In order to satisfy the requirement of being a subscriber to measured data topics, the concept based on callback functions was chosen.

The main function serves firstly to process arguments and to set the corresponding variables. After that it creates an instance of **SharedObjects** class (available in *src/ins.cpp*), which is discussed separately. Almost all the time of the node's life it runs in an endless while loop with only one command – **ros::spin()** – and running condition **ros::ok()**. This means that it will continue processing all the callbacks until the ROS terminates the node.

4.5 Class SharedObjects

This class is implemented in the file *src/ins.cpp*. It contains all the callback functions realizing the data processing:

Functions **GpsCallback** and **OdometryCallback** only log the respective data into their log files at the times the data are published by the **mtig_node**, which was developed by ETH (Die **Eidgenössische Technische Hochschule** Zürich - The Swiss Federal Institute of Technology Zurich) from the base Xsens' node (available in ROS repository). The callback function reacting on data from inertial measurements – **ImuCallback** – performs the INS mechanization and publishes the output data to topics */mechanization_output* and */tf* and their logging into the respective file.

Function **getParameters** loads the user-definable constants into the program.

5 Experimental evaluation and results

The final product of this work – INS mechanization (package ***inso***) is an extension to attitude mechanization implemented in the package ***ins***. That is the reason the testing is divided into two parts.

The first part covers the testing of the ***ins*** node, i. e. attitude mechanization algorithm implementation, the other part covers the testing of the ***inso*** node, which includes the ***ins*** node's algorithms and extends them into full attitude and position mechanization.

5.1 Testing location

The testing data was collected in the lab number 10 in the G building (Center for Machine Perception, Faculty of Electrical Engineering, Czech Technical University in Prague).

The IMU (see Appendix D) was embedded inside the robot by the manufacturer.

5.2 Common testing procedure

All the tests were performed using these steps:

1) Executing *roscore* command to initialize ROS.
2) Launching the nodes using *roslaunch ins ins.launch* or *roslaunch inso inso.launch* command.
3) Using *rosbag play <desired bagfile>* command to play the collected data.
4) After finishing the bagfile playback, the mechanization output file is saved with name *ins_matlab* – the file for post-processing by resource algorithm.
5) Relaunching the node (see step 2) to create a new logging file.
6) Using *rosbag* to play only the first 500 samples of the bagfile (the first 500 samples are used by both algorithms – depending on settings – to perform basic calibration, but MATLAB uses the same data for mechanization cycle, the C++ implementation starts the cycle after the calibration, ignoring the calibration data).
7) Using rosbag to play the whole bagfile again (the first 500 samples are repeated to process all the data with the C++ mechanization cycle).
8) Saving the new file with the name *ins_calib.*
9) Processing *ins_matlab* with resource algorithm, loading *ins_calib* file to MATLAB.
10) Plotting MATLAB implementation results with results saved in *ins_calib* file (plotting realized by MATLAB script – Appendix C, for plotting other data used similar scripts)

5.3 Part 1 – Attitude mechanization (package **ins**)

There were several tests performed to compare the original algorithm with the implemented C++ version. The results of all tests were virtually the same, so only one test's results are presented. Initial yaw Euler angle was set to 0 for both algorithms (C++, MATLAB script).

Movement description:

During this test the robot was manually rotated, firstly about its *x*-axis in both directions, then about its *y*-axis in both directions. After that it was rotated using its own motors around the *z*-axis. Its initial and final orientations were the same.

Measurement results:

Fig. 3 shows the movement described in the previous paragraph. The most important information the graphs contain cannot be seen explicitly: The maximum differences between MATLAB post-processing results and the **ins** node real-time processing results are shown in Table 5.

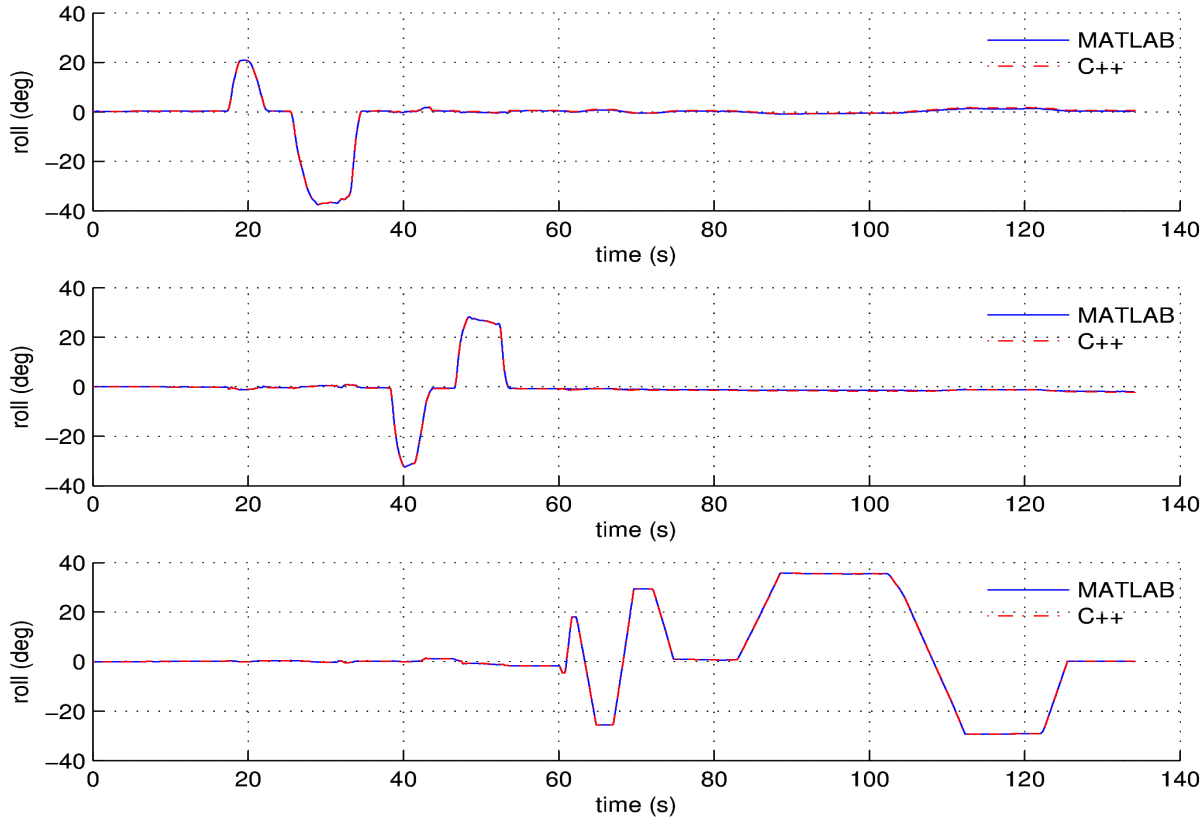


Figure 3: Comparison of the original MATLAB algorithm and C++ implementation (node **ins**)

	t (s)	d (deg)
roll	0.075	-0.1958
pitch	117.5	-0.3728
yaw	117.5	0.0620

Table 5: Maximum differences between MATLAB and C++ implementation

5.4 Part 2 – Attitude and position mechanization (package **inso**)

There were two tests done to evaluate this node. The first of them was realized by moving the robot repeatedly forward and backward to see the characteristic steps in velocity. The other test was longer and it's motion copied an „L“ track there and then back.

5.4.1 Test 1 – Repeated linear motion

Movement description:

The initial yaw angle for both algorithms was set to 0. The robot was moved forward and backward 5 times with each movement about 1 m long, without any rotation.

Measurement results:

The computation of position requires double integration of acceleration. Because of that the 2 figures are shown – one for velocities comparison (Fig. 4) and the other one (Fig. 5) for positions comparison of both algorithms.

The maximum differences between velocities and positions computed by both algorithms are captured in Table 6 and Table 7, respectively.

	t (s)	dv (m/s)
v_N	62.617	-0.5664
v_E	62.617	-0.1787
v_D	4.15	0.004244

Table 6: Maximum velocity differences

	t (s)	dPN (m)
P_N	62.617	-12.3470
P_E	62.617	-3.3192
P_D	62.617	0.0999

Table 7: Maximum position differences

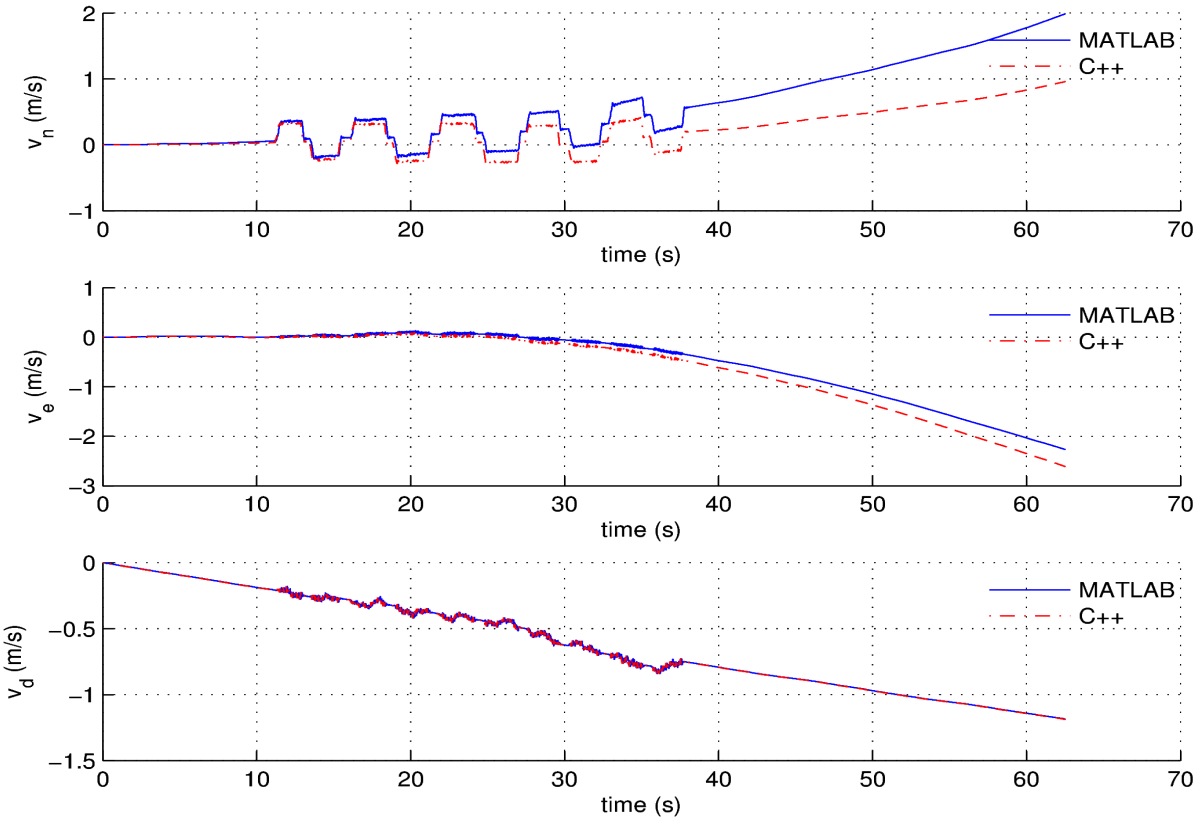


Figure 4: Velocities comparison plots

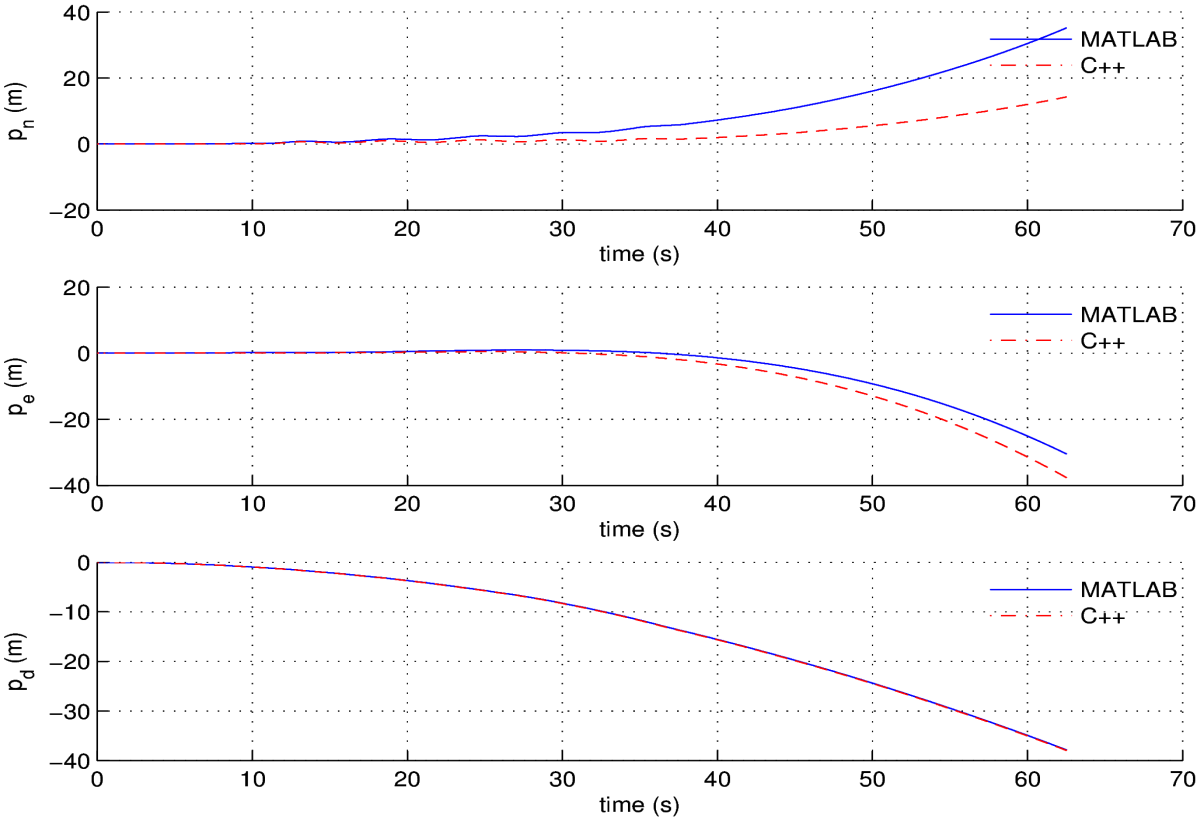


Figure 5: Positions comparison plots

5.4.2 Test 2 – The „L“ motion

Movement description:

As it was said, the robot followed an „L“ path, firstly by moving about 3 m forward, turning 90 degrees left and moving about 5 m forward. Then it followed the same path back.

Measurement results:

There are the same resulting data as in the previous test displayed here. The velocities and positions plots are shown in Fig. 6 and Fig. 7.

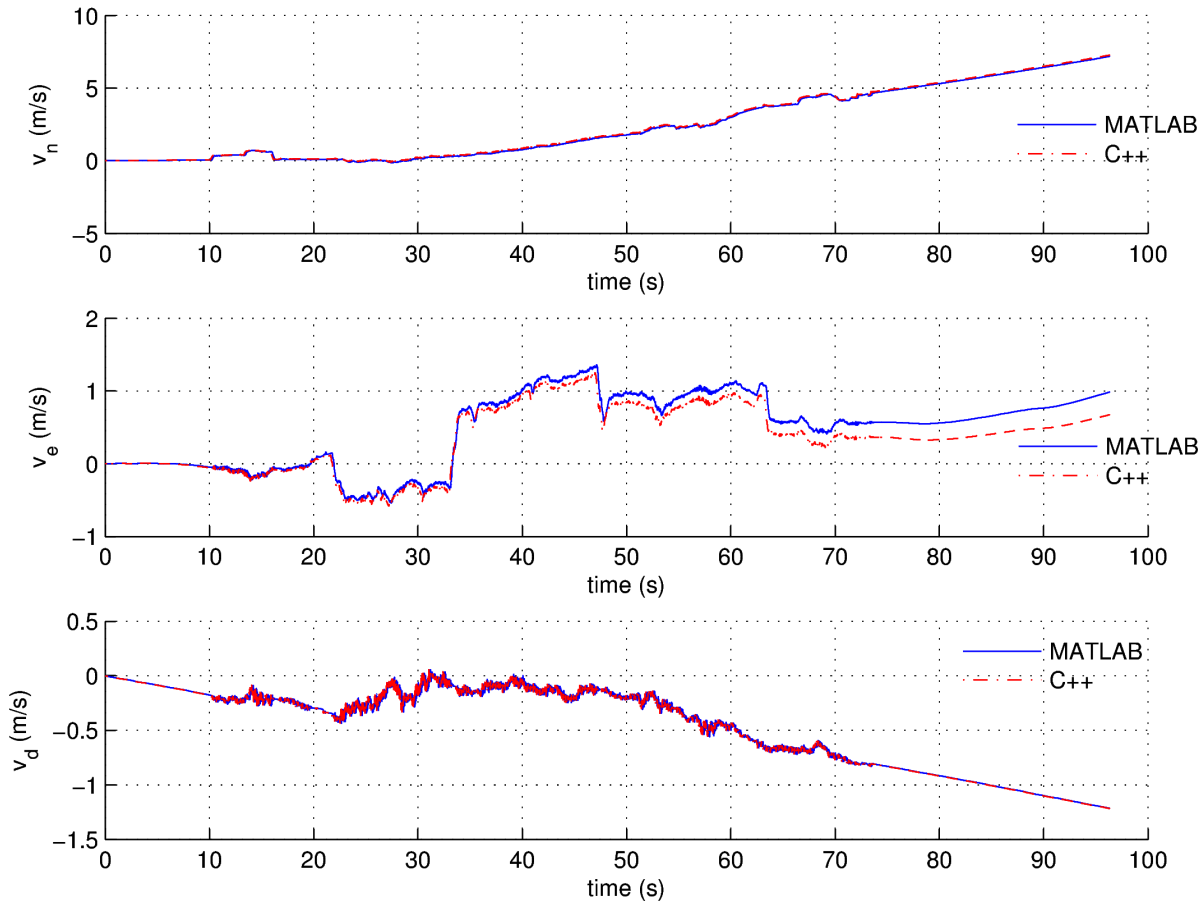


Figure 6: Velocities comparison plots

	t (s)	dv (m/s)
v_N	96.417	-0.2404
v_E	96.417	-0.3161
v_D	96.417	-0.0038

Table 8: Maximum velocity differences

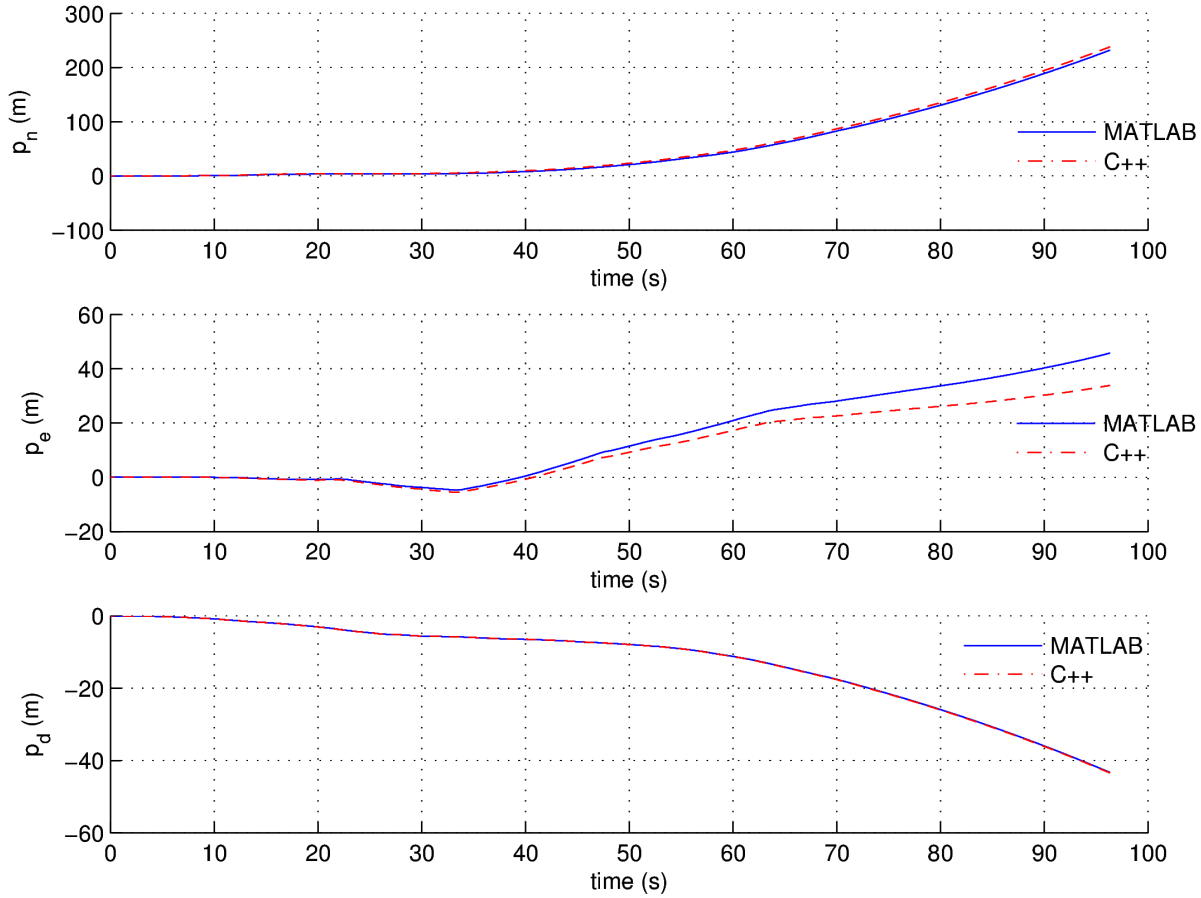


Figure 7: Positions comparison plots

	t (s)	dP (m/s^2)
P_N	96.417	-8.2542
P_E	96.417	-12.9992
P_D	96.417	-0.1941

Table 9: Maximum position differences

6 Conclusion

6.1 Implementation of **ins** and **inso** nodes

The Fig. 3 and Table 5 show minimal differences between the implementations. Thanks to that we can say without discussion, that the **ins** node was implemented successfully.

Regarding the **inso** node, the situation is different. All figures (Fig. 4 - Fig. 7) show that the *z*-axis velocities and positions are almost perfectly identic (except minimal error, caused probably by different MATLAB and C++ numerics), but results of the remaining two axes are very different from the first look at the figures. The question is: Which results are better?

There is only one way how to evaluate the results: All the plots should be static in general, except of desired dynamics caused by the robot's motion. From this point of view the best plot to evaluate the results is Fig. 4. We can see that the dynamics of the velocity in the north direction is the same in both cases, but the drifting of the C++ implementation is much lower, but when we look into the v_e plot the situation is different and the MATLAB implementation seems to be slightly better.

It can't be concluded definitely, which implementation is more accurate. The **inso** node is still in-development product, which will be further extended (with odometry and GPS data fusion with advanced state estimation methods) and eventually debugged.

6.2 Summary of the summer work experience

Although this work is mainly about the conversion of MATLAB post-processing algorithms to C++ real-time implementation, there were many other things that needed to be done in order to successfully work with the robot.

1) Studying ROS documentation was necessary to operate the robot, collect the data, launching nodes (including utilities *rviz*, *rxbag*).
2) Studying OpenCV documentation for successfully implementing both **ins** and **inso** nodes.
3) Hours of experimental evaluation and field-testing , collecting of data, and testing the nodes.

References

[1] P. G. Savage, "Strapdown Inertial Navigation Integration Algorithm Design Part 1: Attitude Algorithms," Journal of Guidance, Control, and Dynamics, vol. 21, no.1, pp. 19 - 28, 1998.

- "Strapdown Inertial Navigation Integration Algorithm Design Part 2: Velocity and Position Algorithms," Journal of Guidance, Control, an Dynamics, vol. 21, no. 2, pp. 208 - 221, 1998

[2] *Willow Garage*. Available from (September 14, 2011): http://www.willowgarage.com/

[3] *ROS.org wiki*. Available from (September 14, 2011): http://www.ros.org/wiki/

Appendix A – *definitions.h* header file

```
#include "cv.h"
#include "ins/mechanization_output.h"

// CONSTANTS
#define   PI          3.14159265358979323846264643
#define   geo_a       6378137.0
#define   geo_e       0.081819191
#define   EULER       2.71828182845904523536028747135266

// ATTITUDE FEEDBACK TYPES
#define   ATT_NO_FEEDBACK                 0
#define   ATT_GYR_ACC_WEIGHTED_AVG        1
#define   ATT_FILTERED                    2

// MEASUREMENT STATES
#define   MEAS_STATE_CALIB                0
#define   MEAS_STATE_CALIB_RESULT         1
#define   MEAS_STATE_FIRST                2
#define   MEAS_STATE_MECHANIZATION        3

#define   YAW_NOT_SET                     1000


using namespace cv;

/*---------------------------------------------------------------
 * TYPES DEFINITIONS
 *---------------------------------------------------------------
 */
    typedef Matx<double, 10, 1> Vec10fCol;
    typedef Matx<double, 7, 1> Vec7fCol;
    typedef Matx<double, 6, 1> Vec6fCol;
    typedef Matx<double, 3, 1> Vec3fCol;

    typedef Matx<double, 4, 1> quat;

    typedef Matx<double, 3, 3> Mat33f;
```

Appendix B - *functions.h* header file

```
#include "definitions.h"

using namespace cv;

double rad2deg(double angleInRadians);
Vec3fCol rad2deg(Vec3fCol angleInRadians);
double deg2rad(double angleInDegrees);
Vec3fCol deg2rad(Vec3fCol angleInDegrees);
Vec3fCol alignment_coarse(double mACCx, double mACCy,
     double mACCz, double mGYRx, double mGYRy, double mGYRz,
     double lat, double g, double er);
Vec3fCol dcm2angle(Mat33f dcm);
Vec3fCol threeaxisrot(double r11, double r12, double r21,
double r31, double r32, double r11a, double r12a);
Vec3fCol comp_gravity(Vec3fCol LLA);
quat rotv2quat(Vec3fCol rotv);
quat rotv2negquat(Vec3fCol rotv);
quat euler2quat(Vec3fCol angles);
Mat33f quat2dcm(quat quat);
Vec3fCol quat2euler(quat q);
quat quatnormalize(quat quat);
quat quatmultiply(quat q, quat r);
double myMean(double *data, int length);
Vec3fCol myCross(Vec3fCol vec1, Vec3fCol vec2);
Mat33f dcmecef2ned(double lat_deg, double lon_deg);
quat quatned2ecef(Vec3fCol LLA);
inso::mechanization_output outputMessage(Vec3fCol eul, quat
quaternion, Mat33f dcm, Vec3fCol pos);
Vec3fCol dcm2latlon(Mat33f dcm);
Mat33f comp_skew(Vec3fCol vector);
Vec3fCol quat2rotv(quat q);
quat quatinv(quat q);
quat quatconj(quat qin);
double lalodist(double la1,double lo1,double la2,double lo2);
double calculateR_N(double LAT);
double calculateR_M(double LAT);
double myNorm(Vec3fCol input);
double myNorm(quat input);
```

Appendix C: *plotScriptEul* MATLAB script

```
function plotScriptEul(data1,data2,sampleRate)
    % make a new figure
    figure ;

    % exclude the calibration period to match data lengths
    % indices 8-10: columns of mechanization-calculated
    % orientation in the output file
    data2 = data2(501:end,8:10);

    % plot all the data to single figure (but 3 plots - each
    % for 1 axis) with time axis values computed using sample
    % rate

    subplot(3,1,1);
    grid on ;
    hold on ;
    plot(linspace(0,(size(data1,2)-...
            1)/sampleRate,size(data1,2)),data1(1,:)) ;

    plot(linspace(0,(size(data2,1)-...
            1)/sampleRate,size(data2,1)),data2(:,1),'r-.') ;

    ylabel('roll (deg)');
    xlabel('time (s)');
    legend('MATLAB','C++');

    subplot(3,1,2) ;
    grid on ;
    hold on ;
    plot(linspace(0,(size(data1,2)-...
            1)/sampleRate,size(data1,2)),data1(2,:)) ;
    plot(linspace(0,(size(data2,1)-...
            1)/sampleRate,size(data2,1)),data2(:,2),'r-.') ;
    ylabel('roll (deg)');
    xlabel('time (s)');
    legend('MATLAB','C++');

    subplot(3,1,3) ;
    grid on ;
    hold on ;
    plot(linspace(0,(size(data1,2)-...
            1)/sampleRate,size(data1,2)),data1(3,:)) ;
    plot(linspace(0,(size(data2,1)-...
            1)/sampleRate,size(data2,1)),data2(:,3),'r-.') ;
    ylabel('roll (deg)');
    xlabel('time (s)');
    legend('MATLAB','C++');
```

Appendix D: Xsens MTi-G technical specification

(page number 5 of „MTi-G leaflet“ available for download at http://www.xsens.com/en/general/mti-g)

MTi-G TECHNICAL SPECIFICATIONS

Attitude and heading

Static accuracy (roll/pitch)	<0.5 deg
Static accuracy (heading)[1]	<1 deg
Dynamic accuracy[2]	1 deg RMS
Angular resolution[3]	0.05 deg
Dynamic range:	
- Pitch	± 90 deg
- Roll/Heading	± 180 deg
Maximum update rate:	
- Onboard processing	120 Hz
- External processing	512 Hz

Position

Accuracy position:	
- SPS	2.5 m CEP
Maximum update rate:	
- Onboard processing	120 Hz
- External processing	512 Hz

Interfacing

Digital interface	RS-232(max 921k6 bps) and USB (ext. converter)
Operating voltage	5 - 30V
Power consumption	610-690 mW (typical) - 910 mW (max)
Interface options I/O	SyncOut, AnalogIn (2x),
GPS antenna	SMA connector, active
Timing accuracy	1 ppm (GPS available)

Maximum operational limits

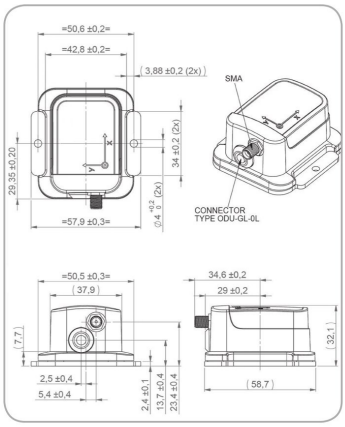
Altitude	18 km
Velocity	515m/s (1854 km/h)
Ambient temperature operating range[4]	-40...+85 °C
Specified performance operating range[4]	0.. +55 °C

INDIVIDUAL SENSOR SPECIFICATIONS

Sensor performance	Rate of turn	Acceleration	Magnetic field	Static pressure
Dimensions	3 axes	3 axes	3 axes	-
Full Scale (standard)	± 300 deg/s	± 50 m/s²	± 750 mGauss	30-120 kPa
Linearity	0.1% of FS	0.2% of FS	0.2% of FS	0.5% of FS
Bias stability[5]	20 deg/h	0.02 m/s²	0.1 mGauss	100 Pa/yr
Scale Factor stability[5]	-	0.03%	0.5%	-
Noise	0.05 deg/s/√Hz	0.002 m/s²/√Hz	1.5 mGauss	4 Pa/√Hz (0.3 m/√Hz)
Alignment error	0.1 deg	0.1 deg	0.1 deg	-
Bandwidth	40 Hz	30 Hz	10 Hz	-
Max update rate	512 Hz	512 Hz	512 Hz	9 Hz

GPS

Receiver type	50 channels L1 frequency, C/A code Galileo L1 Open Service
GPS update rate	4 Hz
Start-up time cold start	29 s
Tracking sensitivity	-160 dBm
Timing accuracy	50 ns RMS



HARDWARE SPECIFICATIONS

Housing

Dimensions (WxLxH)	58x58x33 mm
Weight	68 g

Options

Full scale acceleration:		Full scale rate of turn:	
5g (50 m/s²)	A53	150 deg/s	G15
18g (180 m/s²)	A83	300 deg/s	G35
		1200 deg/s	G25

Product code: MTi-G-28 A## G##
Standard version: MTi-G-28 A53 G35

The MTi-G is RoHS compliant

Note: Specifications subject to change without notice

[1] depends on usage scenario. In case the Earth magnetic field is used, it must be homogeneous
[2] under condition of a stabilized Xsens sensor fusion algorithm and good GPS availability
[3] standard deviation of zero-mean angular random walk
[4] non-condensing environment
[5] deviation over operating temperature range