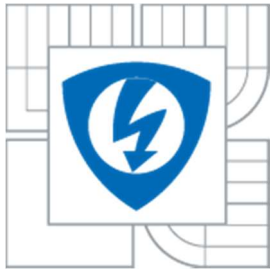




BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF RADIOELECTRONICS

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH
TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

NEURAL NETWORKS IN INERTIAL NAVIGATION SYSTEMS

NEURONOVÉ SÍTĚ V INERCIÁLNÍCH NAVIGAČNÍCH SYSTÉMECH

DOCTORAL THESIS
DOKTORSKÁ PRÁCE

AUTHOR
AUTOR PRÁCE

ING. LENKA TEJMLOVÁ

SUPERVISOR
VEDOUČÍ PRÁCE

DOC. ING. JIŘÍ ŠEBESTA, PH.D.

BRNO 2017

Abstract

The dissertation is focused on inertial navigation systems and dead reckoning positioning. The issue in the problematics is that the dead reckoning systems and inertial navigation systems are inaccurate for medium-term and long-term application due to cumulative errors, assuming that the positioning is not supported by another external system. The dissertation shows possible approaches to the issue of more accurate positioning system based only on the inertial sensors. Basically we are talking about 9-DOF inertial measurement unit that allows sensing the global acceleration, rotation rate and magnetic field strength in three particular axes. The new approach brings artificial neural networks into data processing, where proper neural network is able to recognize the character of motion leading to improvement in positioning. The description of the proposed method includes an analytical procedure of its development and, if possible, the analytical performance assessment. Proposed artificial neural networks are modelled in MATLAB™ and they are used for the determination of the state of the inertial unit. Due to this determination, the position of the inertial measurement unit is evaluated with higher accuracy. An application using Qt framework was developed to create an evaluation system with user interface for standard inertial measurement unit. The designed system based on artificial neural networks was verified by experiments using real sensor data.

Keywords

IMU, INS, DR, inertial positioning, dead reckoning, artificial neural network, Arduino UNO, X-NUCLEO-IKS01A1, MATLAB™, Qt.

Abstrakt

Disertační práce je zaměřena na oblast inerciálních navigačních systémů a systémů, které pro odhad polohy používají pouze výpočty. Důležitým faktem v dané problematice je vysoká nepřesnost určení polohy při střednědobém a dlouhodobém využívání takového systému díky kumulativní chybě za předpokladu, že inerciální systém není podpořen žádným dalším přídavným systémem. V disertační práci jsou uvedeny možné přístupy k této problematice a návrh na zvýšení přesnosti určování polohy pouze na základě inerciálních senzorů. Základem inerciální měřicí jednotky je systém s 9 stupni volnosti, který umožňuje snímat celkové zrychlení, rychlost rotace a sílu magnetického pole, jednotlivě ve třech osách. Klíčovou myšlenkou je zařazení umělých neuronových sítí do navigačního systému tak, že jsou schopny rozpoznat charakteristické rysy pohybů, a tím zvýšit přesnost určení polohy. Popis navrhovaných metod zahrnuje analytický postup jejich vývoje a tam, kde je to možné, i analytické hodnocení jejich chování. Neuronové sítě jsou navrhovány v prostředí MATLAB™ a jsou používány k určení stavu inerciální jednotky. Díky implementaci neuronových sítí lze určit pozici jednotky s řádově vyšší přesností. Aby byl inerciální polohovací systém s možností využití neuronových sítí demonstrativní, byla vyvinuta aplikace v prostředí Qt. Navržený systém a neuronové sítě byly použity při vyhodnocování reálných dat měřených senzory.

Klíčová slova

IMU, INS, DR, inerciální polohovací systém, umělá neuronová síť, Arduino UNO, X-NUCLEO-IKS01A1, MATLAB™, Qt.

Tejmlová, L. *Neural networks in inertial navigation systems*. Dissertation. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, 2016. 129 pages.

Declaration

I declare that I have written my doctoral thesis on the theme of “Neural networks in inertial navigation systems” independently, under the guidance of the doctoral thesis supervisor and using the technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the doctoral thesis I furthermore declare that, as regards the creation of this doctoral thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone’s personal and/or ownership rights and I am fully aware of the consequences in the case of breaking Regulation § 11 and the following of the Copyright Act No 121/2000 Sb., and of the rights related to intellectual property right and changes in some Acts (Intellectual Property Act) and formulated in later regulations, inclusive of the possible consequences resulting from the provisions of Criminal Act No 40/2009 Sb., Section 2, Head VI, Part 4.

Acknowledgement

I would like to express my gratitude to my supervisor doc. Ing. Jiří Šebesta, Ph.D. for his mentoring, consultation, patience and inspiring suggestions throughout my research. I would also like to thank Petr Zelina for inspiring suggestions, long discussions and software support, and my husband Jan Tejml for his endless encouragement and patience throughout my studies.

I would like to thank my parents, Mgr. Marie Zelinová and Ing. František Zelina, for standing beside me throughout my writing this thesis. They have been my inspiration and motivation for continuing to improve my knowledge and move me forward.

CONTENTS

ABSTRACT.....	I
ACKNOWLEDGEMENT.....	III
CONTENTS	IV
LIST OF FIGURES.....	VI
LIST OF TABLES	X
LIST OF ABBREVIATIONS.....	XI
1 INTRODUCTION	1
2 STATE OF THE ART	3
2.1 HISTORY	3
2.2 A DIFFERENT APPROACH	4
2.3 COORDINATE SYSTEMS	4
2.4 SENSOR ERROR MODELS.....	6
2.4.1 <i>List of sensor errors</i>	8
2.5 SENSOR CALIBRATION.....	9
2.6 ORIENTATION DETERMINATION	10
2.6.1 <i>Euler angles</i>	10
2.6.2 <i>Rotation vector</i>	10
2.6.3 <i>Direction cosine matrix</i>	11
2.6.4 <i>Quaternions</i>	12
2.7 ARTIFICIAL NEURAL NETWORK	14
2.8 KALMAN FILTERING.....	15
2.8.1 <i>Implementation</i>	15
2.9 TRAJECTORY RECONSTRUCTION	16
2.10 PROBLEMS	17
3 DISSERTATION OBJECTIVES	18
4 INERTIAL MEASUREMENT UNIT	20
4.1 HARDWARE.....	20
4.2 FIRMWARE	21
4.3 DATA ACQUISITION.....	22
4.4 SENSOR CALIBRATION AND COMPENSATION	25
4.4.1 <i>Bias of the accelerometer</i>	27
4.4.2 <i>Bias of the gyroscope</i>	28
4.4.3 <i>Calibration and compensation</i>	28
5 NEURAL NETWORKS	31
5.1 PRINCIPLE OF ANN.....	32
5.2 ANN TRAINING RESULTS	37
5.2.1 <i>Results of TDNNs with purelin transfer function in output layer</i>	38
5.2.2 <i>Results of TDNNs with tansig transfer function in output layer</i>	46
5.3 ADDITIONAL ANNS	54
5.4 OTHER TYPES OF ANNS.....	55
6 DATA PROCESSING	61
6.1 WHEN THE STATE IS “STILL”	61
6.2 WHEN THE STATE IS “WALKING”	65
6.3 PROBLEMS	67
6.4 FILTERING.....	67

6.4.1	<i>KF applied on ANN outputs</i>	67
6.4.2	<i>Kalman filtering of sensor data</i>	69
6.4.3	<i>Kalman filtering of velocity and position</i>	70
6.4.4	<i>Additional filtration methods</i>	70
6.4.5	<i>Additional filters implementation</i>	71
7	STRAPDOWN NAVIGATION IMPLEMENTATION	73
7.1	BASICS OF STRAPDOWN ALGORITHM	73
7.2	SOFTWARE FOR INERTIAL MEASUREMENT UNIT – TRACKER	73
7.2.1	<i>Receiver</i>	74
7.2.2	<i>Calibrator</i>	75
7.2.3	<i>Derotator</i>	77
7.2.4	<i>Integrator</i>	79
7.2.5	<i>Custom graphs</i>	80
7.2.6	<i>FileHandler</i>	81
7.2.7	<i>Graphical representations in software</i>	81
8	VERIFICATION BY MEASUREMENT	83
8.1	CASE I: IMU STAYED STILL ON THE TABLE	83
8.2	CASE II: IMU STAYED STILL HELD IN THE HAND	85
8.3	CASE III: IMU HELD IN THE HAND DURING STEP AND STOP MOTION	87
8.4	CASE IV: IMU HELD IN THE HAND DURING DISCONTINUOUS WALKING	93
8.5	CASE V: IMU HELD IN THE HAND DURING FLUENT WALKING	94
9	CONCLUSIONS	95
	APPENDIX A COORDINATE TRANSFORMATIONS	97
A.1	REPRESENTATION OF TRANSFORMED VECTORS	97
A.2	UNIT COORDINATE VECTORS	97
A.3	DIRECTION COSINES	97
A.4	RPY/ENU AND RPY/NED TRANSFORMATIONS	98
A.5	ENU/ECEF AND NED/ECEF TRANSFORMATIONS	99
A.6	COMPOSITION OF COORDINATE TRANSFORMATIONS	101
	APPENDIX B MEASUREMENT CONDITIONS	101
B.1	GEOGRAPHIC COORDINATES	101
B.2	TEMPERATURE STABILITY	102
B.3	CALIBRATION PARAMETERS	102
	APPENDIX C HARDWARE SETTING	102
C.1	HARDWARE DESCRIPTION	102
C.2	SENSOR SETTINGS	103
C.3	I ² C COMMUNICATION	104
	APPENDIX D FIRMWARE	105
	APPENDIX E DATA PROCESSING	107
	REFERENCES	110
	OWN PUBLICATIONS	113
	CURRICULUM VITAE	114

LIST OF FIGURES

FIGURE 1.1	RECORDED TRAJECTORY.....	2
FIGURE 2.1	ECI, ECEF, AND GEODETIC COORDINATE FRAME, [11].	5
FIGURE 2.2	DEFINITION OF THE BODY-FIXED FRAME (RPY) WITH RESP. TO LTP FRAME (NED COORDINATES), [12].	6
FIGURE 2.3	INERTIAL NAVIGATION SYSTEM ERROR AS A FUNCTION OF SENSOR ERROR AND TRACKING TIME.....	7
FIGURE 2.4	SENSOR BLACK-BOX MODEL.....	7
FIGURE 2.5	COMMON INPUT-OUTPUT RECOVERABLE ERRORS, IN ORDER FIRST LINE: BIAS, SCALE FACTOR, NONLINEARITY; IN ORDER SECOND LINE: \pm ASYMMETRY, DEAD-ZONE, QUANTIZATION, [13].	9
FIGURE 2.6	ROTATING COORDINATES.	11
FIGURE 2.7	THE ARTIFICIAL NEURON MODEL, [18].	14
FIGURE 2.8	THE STRUCTURE OF ANN, [22].	14
FIGURE 2.9	BASIC CONCEPT OF KALMAN FILTERING, [35].	15
FIGURE 2.10	KALMAN FILTER PROCESS, [35].	16
FIGURE 2.11	SIMPLE STRAPDOWN INS AND ITS OUTPUTS [13].	17
FIGURE 4.1	MINIMU-9 V2 BOARD AND SENSOR AXIS ORIENTATION, [46].	20
FIGURE 4.2	X-NUCLEO-IKS01A1 BOARD [46].	21
FIGURE 4.3	ONE ARDUINO COMMUNICATION CYCLE, CONSEQUENTLY. TX, SDA, SCL.	21
FIGURE 4.4	DIRECTION OF DETECTABLE ANGULAR RATES.	23
FIGURE 4.5	DIRECTION OF DETECTABLE ACCELERATIONS.....	23
FIGURE 4.6	DIRECTION OF DETECTABLE MAGNETIC FIELD.....	23
FIGURE 4.7	BODY-FIXED RPY (ROLL-PITCH-YAW) AXIS.....	23
FIGURE 4.8	SINGLE-SIDED AMPLITUDE SPECTRUM OF THE SENSOR'S DATA – WALK.	24
FIGURE 4.9	SINGLE-SIDED AMPLITUDE SPECTRUM OF THE SENSOR'S DATA – SWINGING.	25
FIGURE 4.10	RAW ROTATION RATE IN TIME WHEN IMU IS STILL – 30 MINUTES.....	25
FIGURE 4.11	THE ERROR IN ANGLE DETERMINATION CAUSED BY OFFSET AND DRIFT.	26
FIGURE 4.12	RAW ACCELERATION AFTER THE EXPECTED VALUE SUBTRACTION WHEN THE IMU IS STILL.....	26
FIGURE 4.13	RAW ROTATION RATES (TOP) AND RAW ACCELERATIONS (BOTTOM) IN PARTICULAR AXIS.....	26
FIGURE 4.14	DATA FROM UNCALIBRATED SENSORS IN TIME [s].	29
FIGURE 4.15	CALIBRATED MAGNETOMETER DATA (360° ROTATION) [MGAUSS] IN TIME [s].	29
FIGURE 4.16	C++ SOURCE CODE – A PART OF CALIBRATION [QT].....	30
FIGURE 4.17	DATA FROM CALIBRATED SENSORS WHILE SHAKING.	31
FIGURE 5.1	NARX ANN PRINCIPLE.	32
FIGURE 5.2	FFTD ANN PRINCIPLE.	32
FIGURE 5.3	LINEAR TRANSFER FUNCTION.....	34
FIGURE 5.4	LOG-SIGMOID TRANSFER FUNCTION.	34
FIGURE 5.5	HYPERBOLIC TANGENT SIGMOID TRANSFER FUNCTION.....	34
FIGURE 5.6	SYMMETRIC HARD-LIMIT TRANSFER FUNCTION.....	34
FIGURE 5.7	MATLAB TM CODE FOR VERIFYING THE SUITABILITY OF TD ANN.	36
FIGURE 5.8	ANN STRUCTURE – FEED FORWARD, TIME DELAY, PURELIN TRANSFER FUNCTION IN OUTPUT LAYER.	37
FIGURE 5.9	ANN STRUCTURE – FEED FORWARD, TIME DELAY, TANSIG TRANSFER FUNCTION IN OUTPUT LAYER.....	37
FIGURE 5.10	TRAINING SET FOR TDNN 1.	38
FIGURE 5.11	TIME-SERIES RESPONSE, EPOCH 125, TDNN_DIVIDEBLOCK_1.....	39
FIGURE 5.12	TIME-SERIES RESPONSE, EPOCH 214, TDNN_DIVIDERAND_1.	39
FIGURE 5.13	TRAINING REGRESSION, EPOCH 125, TDNN_DIVIDEBLOCK_1.	40
FIGURE 5.14	TRAINING REGRESSION, EPOCH 214, TDNN_DIVIDERAND_1.	40

FIGURE 5.15	ERROR HISTOGRAM, EPOCH 125, TDNN_DIVIDEBLOCK_1.	40
FIGURE 5.16	ERROR HISTOGRAM, EPOCH 214, TDNN_DIVIDERAND_1.	40
FIGURE 5.17	TRAINING SET FOR TDNN 2.	41
FIGURE 5.18	TIME-SERIES RESPONSE, EPOCH 999, TDNN_DIVIDEBLOCK_2.	41
FIGURE 5.19	TIME-SERIES RESPONSE, EPOCH 86, TDNN_DIVIDERAND_2.	42
FIGURE 5.20	TRAINING REGRESSION, EPOCH 999, TDNN_DIVIDEBLOCK_2.	42
FIGURE 5.21	TRAINING REGRESSION, EPOCH 86, TDNN_DIVIDERAND_2.	42
FIGURE 5.22	ERROR HISTOGRAM, EPOCH 999, TDNN_DIVIDEBLOCK_2.	42
FIGURE 5.23	ERROR HISTOGRAM, EPOCH 86, TDNN_DIVIDERAND_2.	42
FIGURE 5.24	TRAINING SET FOR TDNN 3.	43
FIGURE 5.25	TIME-SERIES RESPONSE, EPOCH 366, TDNN_DIVIDEBLOCK_3.	44
FIGURE 5.26	TIME-SERIES RESPONSE, EPOCH 81, TDNN_DIVIDERAND_3.	44
FIGURE 5.27	TRAINING REGRESSION, EPOCH 366, TDNN_DIVIDEBLOCK_3.	44
FIGURE 5.28	TRAINING REGRESSION, EPOCH 81, TDNN_DIVIDERAND_3.	44
FIGURE 5.29	ERROR HISTOGRAM, EPOCH 81, TDNN_DIVIDERAND_3.	45
FIGURE 5.30	ERROR HISTOGRAM, EPOCH 366, TDNN_DIVIDEBLOCK_3.	45
FIGURE 5.31	GRAPHICAL COMPARISON OF TRAINED FEED FORWARD TIME DELAYED NETWORKS.	46
FIGURE 5.32	TRAINING SET FOR TDNN 1 TS.	46
FIGURE 5.33	TIME-SERIES RESPONSE, EPOCH 29, TDNN_DIVIDEBLOCK_1_TS.	47
FIGURE 5.34	TIME-SERIES RESPONSE, EPOCH 27, TDNN_DIVIDERAND_1_TS.	47
FIGURE 5.35	TRAINING REGRESSION, EPOCH 29, TDNN_DIVIDEBLOCK_1_TS.	48
FIGURE 5.36	TRAINING REGRESSION, EPOCH 27, TDNN_DIVIDERAND_1_TS.	48
FIGURE 5.37	ERROR HISTOGRAM, EPOCH 29, TDNN_DIVIDEBLOCK_1_TS.	48
FIGURE 5.38	ERROR HISTOGRAM, EPOCH 27, TDNN_DIVIDERAND_1_TS.	48
FIGURE 5.39	TRAINING SET FOR TDNN 2 TS.	49
FIGURE 5.40	TIME-SERIES RESPONSE, EPOCH 28, TDNN_DIVIDEBLOCK_2_TS.	49
FIGURE 5.41	TIME-SERIES RESPONSE, EPOCH 28, TDNN_DIVIDERAND_2_TS.	50
FIGURE 5.42	TRAINING REGRESSION, EPOCH 28, TDNN_DIVIDEBLOCK_2_TS.	50
FIGURE 5.43	TRAINING REGRESSION, EPOCH 28, TDNN_DIVIDERAND_2_TS.	50
FIGURE 5.44	ERROR HISTOGRAM, EPOCH 28, TDNN_DIVIDEBLOCK_2_TS.	50
FIGURE 5.45	ERROR HISTOGRAM, EPOCH 28, TDNN_DIVIDERAND_2_TS.	50
FIGURE 5.46	TRAINING SET FOR TDNN 3 TS.	51
FIGURE 5.47	TIME-SERIES RESPONSE, EPOCH 90, TDNN_DIVIDEBLOCK_3_TS.	52
FIGURE 5.48	TIME-SERIES RESPONSE, EPOCH 73, TDNN_DIVIDERAND_3_TS.	52
FIGURE 5.49	TRAINING REGRESSION, EPOCH 90, TDNN_DIVIDEBLOCK_3_TS.	53
FIGURE 5.50	TRAINING REGRESSION, EPOCH 73, TDNN_DIVIDERAND_3_TS.	53
FIGURE 5.51	ERROR HISTOGRAM, EPOCH 90, TDNN_DIVIDEBLOCK_3_TS.	53
FIGURE 5.52	ERROR HISTOGRAM, EPOCH 73, TDNN_DIVIDERAND_3_TS.	53
FIGURE 5.53	GRAPHICAL COMPARISON OF TRAINED FEED FORWARD TIME DELAYED NETWORKS.	54
FIGURE 5.54	THE DATA USED AS A PART OF THE TRAINING SET FOR ADD_TDNN_2 TRAINING.	55
FIGURE 5.55	RESPONSE OF THE ADD_TDNN_2.	55
FIGURE 5.56	NARX TD ANN STRUCTURE.	56
FIGURE 5.57	NARX TD ANN RESPONSE.	56
FIGURE 5.58	TRAINING PARAMETERS FOR NARX TD ANN.	56
FIGURE 5.59	TRAINING REGRESSION, EPOCH 42.	56
FIGURE 5.60	FF ANN STRUCTURE (WITHOUT TIME DELAY).	57
FIGURE 5.61	FF ANN RESPONSE (WITHOUT TIME DELAY).	57
FIGURE 5.62	TRAINING PARAMETERS FOR FF ANN.	57
FIGURE 5.63	TRAINING REGRESSION, EPOCH 67.	57
FIGURE 5.64	NARX ANN STRUCTURE (WITHOUT TIME DELAY).	58

FIGURE 5.65	NARX ANN RESPONSE (WITHOUT TIME DELAY).....	58
FIGURE 5.66	TRAINING PARAMETERS FOR FF ANN.	58
FIGURE 5.67	TRAINING REGRESSION, EPOCH 74.	58
FIGURE 5.68	PR&C ANN STRUCTURE (3 HIDDEN NEURONS).	59
FIGURE 5.69	TRAINING PARAMETERS PR&C ANN.	59
FIGURE 5.70	CONFUSION MATRIX I.	59
FIGURE 5.71	TRAINING PARAMETERS PR&C	60
FIGURE 5.72	CONFUSION MATRIX II.	60
FIGURE 6.1	DEROTATION FROM THE RPY (BODY) COORDINATES TO ENU COORDINATES.....	62
FIGURE 6.2	QUATERNION ORIENTATION DETERMINATION.	63
FIGURE 6.3	A PART OF C-CODE FOR DEROTATION.....	64
FIGURE 6.4	SINGLE STEP OF SUCCESSIVE ATTITUDE DETERMINATION.	65
FIGURE 6.5	IMPROVED OUTPUT OF THE ARTIFICIAL NEURAL NETWORK (FILTERED BY LKF).	69
FIGURE 6.6	KALMAN FILTER APPLIED ON THE ACCELERATION DATA.	69
FIGURE 6.7	KALMAN FILTER APPLIED ON THE ROTATION RATE SENSOR DATA.....	70
FIGURE 6.8	RAW PARTICULAR ROTATIONS IN TIME WHEN IMU IS STILL – 30 MINUTES.....	71
FIGURE 6.9	ACCELERATION DRIFT IMPACT.	72
FIGURE 7.1	RAW DATA.....	75
FIGURE 7.2	CALIBRATED DATA.	76
FIGURE 7.3	SETTING DIALOG WINDOW FOR TRACKER.	78
FIGURE 7.4	DEROTATED DATA.	78
FIGURE 7.5	INTEGRATED DATA.....	80
FIGURE 7.6	EXAMPLE OF CUSTOM PLOT DATA.....	80
FIGURE 7.7	A DIALOG WINDOW FOR RECORDING AND REPLAYING THE DATA.	81
FIGURE 7.8	THE SOFTWARE CONTROLS.	82
FIGURE 7.9	VISUAL INFORMATION ABOUT THE HEADING AND THE STATE	82
FIGURE 7.10	CUBE VIEW	82
FIGURE 8.1	EVALUATED TRAJECTORY IN TIME, THE UOG IN [M] (LEFT Y-AXIS) AND OTHERS IN [MM] (RIGHT Y-AXIS), STILL ON THE TABLE.	85
FIGURE 8.2	EVALUATED TRAJECTORY IN TIME, THE UOG IN [M] (LEFT Y-AXIS) AND OTHERS IN [M] (RIGHT Y-AXIS)	87
FIGURE 8.3	DETAILED POSITIONS OF STEP AND STOP MEASUREMENT.	87
FIGURE 8.4	SILLYSTATUS STATE (RED) AND ANN OUTPUT STATE IN TIME [s].	88
FIGURE 8.5	ENU VELOCITY ESTIMATED IN UOG MODE, X-AXIS REPRESENTS MEAS. TIME IN [s].	89
FIGURE 8.6	ENU VELOCITY ESTIMATED WITH SILLYSTATUS FILTER IN PROCESS, X-AXIS REPRESENTS MEAS. TIME IN [s].	89
FIGURE 8.7	ENU VELOCITY ESTIMATED WITH ANN IN PROCESS, X-AXIS REPRESENTS MEAS. TIME IN [s].	89
FIGURE 8.8	ENU POSITION ESTIMATED IN UOG MODE, X-AXIS REPRESENTS MEAS. TIME IN [s].	89
FIGURE 8.9	ENU POSITION ESTIMATED WITH SILLYSTATUS FILTER IN PROCESS, X-AXIS REPRESENTS MEAS. TIME IN [s].	90
FIGURE 8.10	ENU POSITION ESTIMATED WITH ANN IN PROCESS, X-AXIS REPRESENTS MEAS. TIME IN [s].	90
FIGURE 8.11	PARTICULAR DISTANCES FROM (0,0,0) POSITION IN ENU COORDINATES.	92
FIGURE 8.12	TRAJECTORY IN ENU COORDINATES IN 2D [M].	92
FIGURE 8.13	TRAJECTORY IN ENU COORDINATES IN 3D [M].	92
FIGURE 8.14	ENU VELOCITY ESTIMATED WITH ANN IN PROCESS [MS ⁻¹] IN TIME [s] - INTERRUPTED WALKING.....	93
FIGURE 8.15	ENU POSITION ESTIMATED WITH ANN IN PROCESS [M] IN TIME[s] - INTERRUPTED WALKING.	93
FIGURE 8.16	TRAJECTORY IN ENU COORDINATES, 3D IN [M] - INTERRUPTED WALKING.	94
FIGURE 8.17	LINEAR ACCELERATION IN ENU COORDINATES IN [MS ⁻²] IN TIME [s], CONTINUOUS WALKING.	94
FIGURE 8.18	POSITION IN ENU COORDINATES IN [M] IN TIME [s], CONTINUOUS WALKING.	94
FIGURE 8.19	COMBINED SILLYSTATUS STATE AND ANN DECISION ON THE IMU STATE (GREEN) IN TIME.	94
FIGURE B.1	CHANGE IN THE MEASURED ROTATION RATE [DEG/S] WHEN THE TEMPERATURE DROPS DOWN (IN 550 s) IN TIME [s].	102
FIGURE C.2	FOUR ARDUINO COMMUNICATION CYCLES, CONSEQUENTLY. TX, SDA, SCL	104

FIGURE C.3	DETAIL OF RS232 COMMUNICATION CYCLE, CONSEQUENTLY. TX, SDA, SCL	104
FIGURE C.4	DETAIL OF I ² C COMMUNICATION CYCLE, CONSEQUENTLY. TX, SDA, SCL	105
FIGURE C.5	TRANSMISSION OF ONE BYTE VIA I ² C. SDA, SCL	105
FIGURE E.6	RAW ACCELERATION [G] IN TIME [S].....	107
FIGURE E.7	CALIBRATED ACCELERATION [G] IN TIME [S].....	107
FIGURE E.8	ENU (LINEAR) ACCELERATION [M·S ⁻²] IN TIME [S].....	107
FIGURE E.9	ENU (LINEAR) ACCELERATION AFTER KF [M·S ⁻²] IN TIME [S].	107
FIGURE E.10	RAW ROTATION RATE [DEG·S ⁻¹] IN TIME [S].	108
FIGURE E.11	CALIBRATED ROTATION RATE [DEG·S ⁻¹] IN TIME [S].....	108
FIGURE E.12	RAW MAGNETIC FIELD [GAUSS] IN TIME [S].	108
FIGURE E.13	CALIBRATED MAGNETIC FIELD [GAUSS] IN TIME [S].....	108
FIGURE E.14	EULER ANGLES [DEG] IN TIME [S].....	109
FIGURE E.15	VELOCITY [M·S ⁻¹] IN TIME [S].....	109
FIGURE E.16	POSITION [M] IN TIME [S].	109
FIGURE E.17	STATE AND ARTIFICIAL NEURAL NETWORK OUTPUT IN TIME [S].....	109

LIST OF TABLES

TABLE 4.1	MAGNETIC FIELD VALUES IN KOHOUTOVICE, BRNO [45].	23
TABLE 4.2	MAGNETIC DECLINATION FOR KOHOUTOVICE, BRNO [47].	29
TABLE 5.1	DEFAULT TRAINING PARAMETERS [MATLAB™].	33
TABLE 5.2	COMPARISON OF NETWORKS – TRAINING PARAMETERS.	45
TABLE 5.3	COMPARISON OF NETWORKS II – TRAINING PARAMETERS.	53
TABLE 5.4	TRAINING PARAMETERS OF THE ADDITIONAL NETWORKS.	54
TABLE 5.5	TRAINING PARAMETERS OF THE OTHER TYPES OF ANNS.	60
TABLE 6.1	BIAS OFFSET AND STANDARD DEVIATION OF RAW ROTATIONS WHEN THE IMU IS STILL.	71
TABLE 7.1	CONSEQUENCES OF DEFINED STATE IN PARTICULAR MODES.	79
TABLE 7.2	VARIABLES AVAILABLE IN CUSTOM PLOT VIEW.	81
TABLE 8.1	VELOCITY AND POSITION, UOG MODE STILL ON THE TABLE.	84
TABLE 8.2	VELOCITY AND POSITION, SILLYSTATUS FILTER USED ONLY STILL ON THE TABLE.	84
TABLE 8.3	VELOCITY AND POSITION, ANN APPLIED STILL ON THE TABLE.	84
TABLE 8.4	VELOCITY AND POSITION, UOG MODE STILL IN THE HAND.	86
TABLE 8.5	VELOCITY AND POSITION, SILLYSTATUS FILTER USED ONLY STILL IN THE HAND.	86
TABLE 8.6	VELOCITY AND POSITION, ANN APPLIED STILL IN THE HAND.	86
TABLE 8.7	POSITIONS OF STEP AND STOP MEASUREMENT.	88
TABLE 8.8	ESTIMATED OUTPUTS OF THE SYSTEM IN PARTICULAR MODES.	90

LIST OF ABBREVIATIONS

ABS	anti-lock brake system
AHRS	attitude and heading reference system
AKBS	adaptive knowledge based system
ANN	artificial neural network
ASR	anti-slip regulation
AUV	autonomous underwater vehicle
BK	bank
BT	bluetooth
CPU	central processing unit
DCM	direction cosine matrix
dly	delay
DOF	degrees of freedom
DOP	dilatation of precision
dps	degrees per second
DR	dead reckoning
E, W, N, S	East, West, North, South
ECEF	Earth-centred, earth-fixed (coordinates)
ECI	Earth-centred inertial (coordinates)
EL	elevation
ENU	east-north-up
FF	feed forward
FFTD	feed-forward time-delay
FIFO	first in first out
FL	fuzzy logic
g	standard acceleration due to gravity (of free fall), 9.80665 ms^{-2}
GLONASS	space-based satellite navigation system used by the Russian
GNSS	global navigation satellite system
GPS	global positioning system
GUI	graphical user interface
H	heading
HPF	high-pass filter
I ² C	inter-integrated circuit
IDE	integrated development environment
IMU	inertial measurement unit
INS	Inertial navigation system
ISA	inertial sensor assembly
KF	Kalman filter
LKF	linear Kalman filter
LPF	low-pass filter
LQE	linear quadratic estimation
LSB	least significant bit
LTP	local tangent plane
LVN	land vehicle navigation

MATLAB™	software, multi-paradigm numerical computing environment
MCU	microcontroller unit
mdps	mili-degrees per second
MEMS	micro-electro-mechanical systems
MinIMU	inertial measurement unit with L3GD20 gyro,
MSE	mean squared error
NA	not available
NaN	not a number
NARX	nonlinear autoregressive network with exogenous inputs
NED	north-east-down
nnum	number of neurons in hidden layer of ANN
NOAA	National Oceanic and Atmospheric Administration
ODR	octal data rate
ORI	orientation of our IMU; defined in quaternion form
PC	personal computer
PCB	printed circuit board
PR&C	pattern recognition and clustering
QT	software, cross-platform application framework
RAM	random access memories
ROT	rotation coordinates
RPY	roll-pitch-yaw
SAE	society of automotive engineers
SCL	serial clock line
SDA	synchronous data
SPS	samples per second
SSD	solid-state drive
TDNN	time-delayed artificial neural network
TX	transmit
U, D	up, down
UOG	use only gyro mode
USB	universal serial bus
WI-FI	wireless fidelity
WMM2015	World Magnetic Model 2015

1 INTRODUCTION

An implementation of artificial intelligence into an automatic navigation systems is the one of opportunities how to improve performances of autonomous positioning systems. Well known positioning method which is used in many modern systems, such in cars, is the dead reckoning. This method is defined as the process of calculating current position by using a previous determined position and actual data from inertial sensors in combination with vehicle odometers. The implementation of this method defines actual position of moving object regarding to the initial position. It also defines the trajectory during the movement.

This topic is often discussed nowadays and the research in this field can be divided to many way. To providing of more effective solutions than independent processing of inertial sensor data offers, additional methods, systems and devices are required.

Research teams work on acquisition with intention to obtain more precise results provided by sensor data fusion, by increasing the number of sensors that are used to measure the same physical quantities, by adding various specific devices, such as Wi-Fi or other wireless equipment and its signal strength, by limitation of results determination, by monitoring of regularities in motions and finally by fusion with available GNSS/GPS, pedestrian navigation constrains, visual-aided constrains, map matching etc.

There are three main issues arising from the fundamentals of inertial navigation. The first of all is the Earth's gravity. We can measure the acceleration. It contains both, a linear acceleration (that is needed to determine the position) and Earth's gravity acceleration. This is good when the accelerometers are placed horizontally (flat). The precise strength of Earth's gravity varies depending on location, nevertheless, at the Earth's surface the nominal average value (standard acceleration of free fall) should be in our case subtracted, because we are located on the Earth's surface. The accelerometers are generally never horizontally placed though the position of inertial measurement unit is often approaching this state. For that reason it is very hard to separate Earth's gravity and linear acceleration, both measured together by accelerometer. We highly focus on this issue in this document. The second difficulty is Earth's rotation around its axis by 15 degrees per hour and around the sun by 0.041 degrees per hour. This should be solved by using the gyrocompass and by implementation of proper compensations in computations. The third issue is a significant inaccuracy caused by sensitivity and typical characteristics of inertial sensors. Due to low signal to noise rate when the linear acceleration of the IMU or its orientation vary is the only inertial sensor navigation fundamentally inapplicable for precise localization. Thus nowadays, many localization methods are combined.

When we are talking about inertial sensor data fusion we are always confronted by real world challenges. It is thought that nothing is exactly accurate and therefore we have to consider deviations and errors as an inseparable part of technique. The task is to use knowledge and enrich it by our own thoughts that complexly lead to invention of better solution, innovation. The application of inertial sensor data fusion brings thorough considerations of error models and their implementation in calculations. In combination with artificial neural network, Kalman filtering and with the support of

GNSS/GPS the dead reckoning system may achieve a sufficient accuracy to determine the orientation and the position where the inertial navigation system (INS) is located.

An inertial navigation system (INS) is a navigation aid that uses a computer, motion sensors (accelerometers) and rotation sensors (gyroscopes, gyros) and maybe others to continuously calculate via dead reckoning (DR) actual position, actual orientation, and actual velocity (direction and speed of movement) of a moving object in time without any external references [1]. It has been called “Newtonian navigation” because its theoretical foundations have been known since time of Newton:

Given the position $x(t_0)$ and velocity $v(t_0)$ of a vehicle at time t_0 , and its acceleration $a(s)$ for times $s > t_0$, then its velocity, $v(t)$, and position, $x(t)$, for all time $t > 0$ can be defined as (2.1.1), (2.1.2).

$$v(t) = v(t_0) + \int_{t_0}^t a(s) ds \quad (2.1.1)$$

$$x(t) = x(t_0) + \int_{t_0}^t v(s) ds \quad (2.1.2)$$

Then, for practical implementation, there are four included issues that have to be solved. Then the result might look like in Figure 1.1.

1. Sensors for measuring acceleration with sufficient accuracy:
 - a. 3-axis acceleration sensor (accelerometer)
 - b. 3-axis rotation sensor (gyroscope)
2. Compatible methods based on integration of the sensor outputs to obtain position
 - a. Methods integrating the gyro outputs to determine the orientation of the accelerometer
 - b. Methods integrating the accelerations to obtain the velocities and integrating the velocities to obtain the position
3. Hardware and software implementing these methods and for interpretation of the results
4. Applications that could justify the investments in technology required for developing the solutions to the capabilities listed above

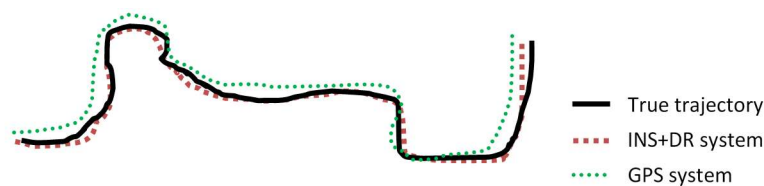


Figure 1.1 Recorded trajectory.

This dissertation thesis heads with the state of the art, where the short history and current research state is described. The next Chapter outlines the objectives of the thesis and the rest of document deals with those objectives. At the end of the main document the results are discussed and the proposed method is evaluated. Annexes complement the described methods.

2 STATE OF THE ART

This Chapter is devoted to the history of the inertial positioning and navigation and related progressive methods, new approaches and methods used in this systems, and particular issues that must be solved when this new approaches are practically applied.

2.1 History

Inertial navigation is a self-contained navigation technique in which measurements provided by accelerometers and gyroscopes are used to track the position and orientation of an object relative to a known starting point, orientation and velocity. Inertial measurement units (IMUs) typically contain three orthogonal rate-gyroscopes and three orthogonal accelerometers, measuring angular velocity and linear acceleration respectively, [2].

By processing signals from these devices, it is possible to track the position and orientation of the device. This aim is often discussed nowadays and research is divided into many directions. To ensure better solution than which is offered by independent processing of sensor data, additional methods and equipment are required. Proposed inertial guidance system is based on dead reckoning method supplemented by artificial neural network (ANN) and Kalman filters (KF).

In 1997, the model based on self-diagnosis system for autonomous underwater vehicles using artificial neural networks was introduced in Advanced Intelligent Mechatronics '97, International Conference. The dynamic model was constructed by an ANN taking advantage of its learning capability. When the sensors seemed to be defective, DR using its corresponding output attempted to scope with the defect. Then the proper action scheme, without extra sensors for the detection, was used to minimize the damage to the autonomous underwater vehicles (AUV), [3].

In 2002, the dead reckoning system in combination with terrain-aided positioning was tested up. The processing is based on multiple Kalman filters that estimate the linear part of the state vector. The appropriate filter is then used for the estimation of remaining part and the simulations showed that the computational load was significantly reduced [4].

The neural network implementation was used by researchers when INS and GPS were integrated. The neural networks for land vehicle navigation (LVN) application was introduced in 2002, [5] and an adaptive neuro-fuzzy model for bridging GPS outages in MEMS-IMU/GPS LVN was introduced in 2004, [6].

An idea to implement ANNs to the personal navigator was presented in Intelligent Signal Processing symposium in 2007. The system integrated GPS, tactical grade IMU, digital barometer, magnetometer and human pedometer to support navigation and tracking of military and rescue system for ground personnel has been developed [7]. One year later, in 2008, the prototype of personal navigator had been developed at The Ohio State University Satellite Positioning and Inertial Navigation (SPIN) Laboratory [8]. The adaptive knowledge based system (AKBS) was based on ANN and Fuzzy Logic (FL) and was trained a priori using sensors data collected by various operators in various environments during GPS signal reception. The KF was then used to improve

the heading information and to reconstruct the trajectory based on step length and step direction.

In 2011 researchers used ANN to the compensation of the reckoning error for AUV when KF were used to estimate positional errors in a soft computing-based models. They evaluated DR error as an output parameter, while attitude angles, velocities and relative time were given as input variables. They validated their algorithms with the conclusion: The absolute error non-compensated is 100.3 m, and the absolute error compensated is 14.8 m. They compared results with GPS position, the voyage was 2800 m with the depth of 0.5 m. This result seems very good however the data measured over this voyage had been used as a training set for to getting of error predictive models [9].

A very nice example of ANNs implementation for navigation has been shown in the paper from 12th International Conference on Control, Automation and Systems, Jeju Island, Korea [10]. Authors of this paper developed the indoor navigation system based on pedestrian dead reckoning (PDR) that uses various sensors in a smartphone. MEMS IMU was mounted on the waist, using sensors and ANN status; they estimated the step length adaptively. They used a map-matching method in addition. If the estimated trajectory was tracked wrong way or the estimated position in unavailable place to go, map matching arranged the estimated position to the coordinate defined in a map. So the computed position was “snapped” to link in the map or to the corner when rotation rate measured by a gyroscope increased in the moment. A barometer was used for to distinguish the floor where the IMU belongs.

A major disadvantage of this method is that we need a map of the area where such a system is used. Without a map, the performance of positioning is not sufficiently accurate.

2.2 A different approach

The presented method approaches to the issue from another point of view than previous solutions of PDR inertial units. It is based on the fact that we need to apply DR (INS) while the terrain is unknown; *that means wireless connections are not available, terrain map is not defined, and GNSS signal is not available*. It was investigated that sensor errors, deviations and drifts achieve significant values, thus, the error in positioning is large. The fusion of sensor data, Kalman filtering and artificial neural network offer a solution for the purpose.

2.3 Coordinate systems

The coordinates for inertial systems are given to be natural to the problem at hand. We use LTP (local tangent plane) coordinates; first-order model of the earth as being flat, where they serve as local reference directions for representing vehicle attitude and velocity for operation – on the surface of the earth (or very close to). A common orientation for LPT coordinates has one horizontal axis (the north axis) in the direction of increasing latitude and the other horizontal axis (the east axis) in the direction of increasing longitude.

Furthermore, we have to specify the ECI (earth-centered inertial) coordinates that are the favoured inertial coordinates in the near-earth environment. The origin of ECI coordinates is at the centre of gravity of the earth, with axis directions:

- x – the direction of the vernal equinox;
- z – parallel to the rotation axis of the earth (north polar axis);
- y – an additional axis to make a right-handed orthogonal coordinate system.

The equatorial plane of the earth is also the equatorial plane of ECI coordinates, nevertheless the earth itself is rotating relative to the vernal equinox by about 15.04109 deg per hour¹. ECEF (Earth-Centred, Earth-Fixed) coordinates have the same origin and third, polar axis as ECI coordinates, but rotate with the earth. Consequently, ECI and ECEF longitudes differ only by a function of time. ECI (indexed by “I”), ECEF (indexed by “e”) coordinates and LTP are shown in Figure 2.1.

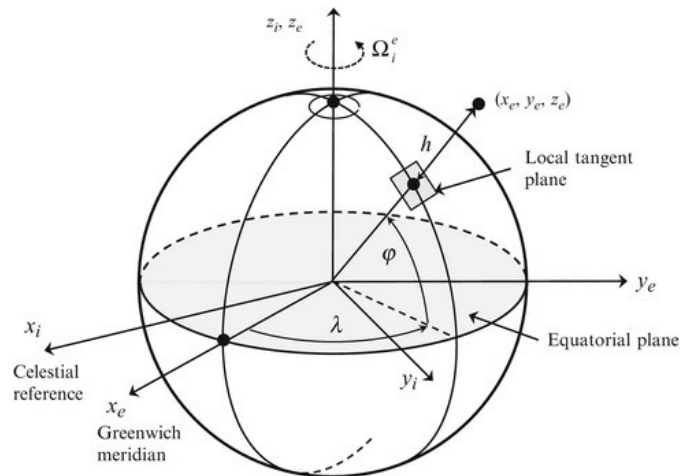


Figure 2.1 ECI, ECEF, and geodetic coordinate frame, [11].

In coordinate system NED (earth-fixed, north-east-down) right-handed LTP system is preferred because the direction of a right (clockwise) turn is in the positive direction with respect to a downward axis and NED coordinate axes coincide with vehicle-fixed RPY (body-fixed, roll-pitch-yaw) coordinates when the vehicle is in the flat position and headed to north. The other, commonly used right-handed LPT system is ENU (east-north-up) and the transformation matrix between ENU and NED shows relation (2.3.1). The ENU coordinate system is preferred in this thesis. The relation between ECEF coordinate frame and ENU coordinate frame can be found in APPENDIX A, part A.5.

RPY coordinates are vehicle fixed, as noted above, with the roll axis in the nominal direction of motion of the vehicle, the pitch axis out the right-hand side, and the yaw axis such that tight turning is positive. This is used also for surface ships and ground vehicles, called SAE coordinates.

¹ *World Book Encyclopaedia* Vol 6. Illinois: World Book Inc.: 1984: 12.
 "It takes 23 hours 56 minutes 4.09 seconds for the Earth to spin around once 2π radians/86164.09 seconds"

In Figure 2.2, body-fixed reference system (RPY coordinates) with respect to LTP frame (NED coordinates) is shown. Other transformations between mentioned coordinate systems are indicated in APPENDIX A.

$$C_{NED}^{ENU} = C_{ENU}^{NED} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad (2.3.1)$$

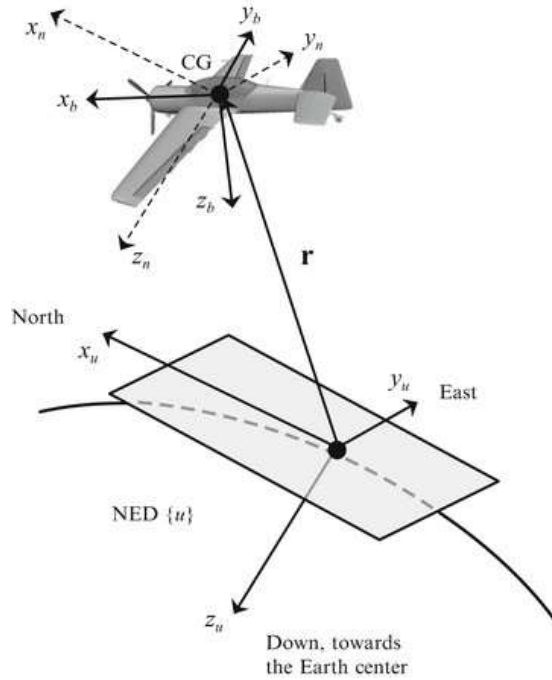


Figure 2.2 Definition of the body-fixed frame (RPY) with resp. to LTP frame (NED coordinates), [12].

2.4 Sensor error models

Inertial navigation performance is hardly limited by the performance of used inertial sensors. The basic formula, Newton's model, gives us an overview of the inertial navigation system's error evolution over time (2.4.1). This is also shown in Figure 2.3 and you can see that the performance significantly decreases with the time and the system based only on integrated data from sensors is inapplicable.

$$\delta_{position} = \frac{1}{2} \delta_{acceleration} \cdot t^2 \quad (2.4.1)$$

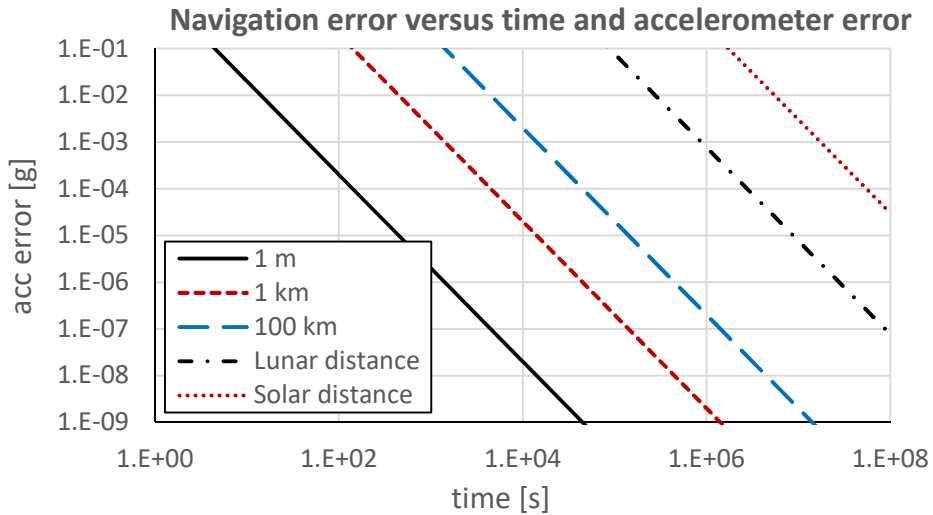


Figure 2.3 Inertial navigation system error as a function of sensor error and tracking time.

The errors in measurement arise from many various reasons. Inertial navigation has been called “black box navigation” because it is entirely self-contained. It interferes what is going on outside by what it can sense inside. In addition, inertial sensors are called black boxes for the same reason. There are more events outside the sensor than just accelerations or rotations², see Figure 2.4.

An important fact to be aware is that accelerometers do not measure gravitational acceleration, but inertial acceleration. That means, they measure “specific force” $a=F/m$, where F is the physically applied force and m is the mass it is applied to [13].

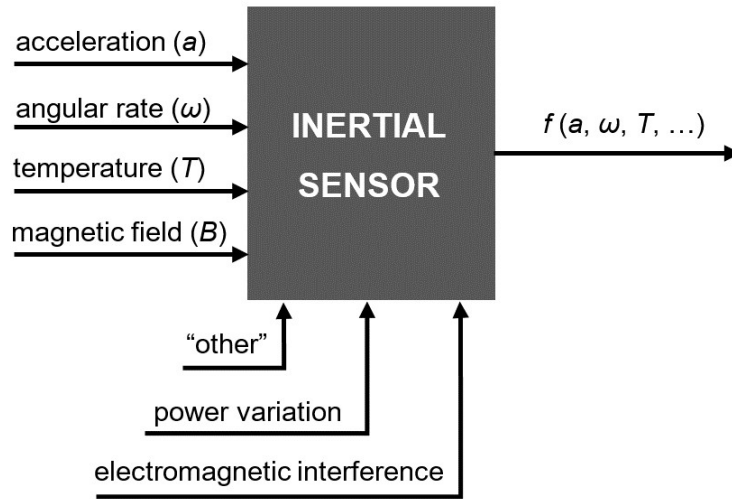


Figure 2.4 Sensor black-box model.

² A comment often heard from inertial sensor designers is „No matter what sort of sensor we design; it always turns out to be a highly sensitive thermometer!” [13].

2.4.1 List of sensor errors

Zero-mean random errors

Standard types of error models from Kalman filtering of zero-mean random errors are used for modelling the unpredictable outputs.

White sensor noise

That noise is generated by power supplies, intrinsic noise in semiconductor devices, or from quantization errors in digitization. It is usually assigned to “electronic noise”.

Exponentially correlated noise

As a time varying additive noise source, the temperature sensitivity of sensor bias driven by ambient temperature variations or by internal heat variations.

Random walk sensor errors

Those errors depends on variances that grow linearly with time and power spectral densities that fall off as $1/f^2$, that means 20 dB per decade, where f is the frequency. Magnitudes of those noises are reflected to the outputs of sensors equivalently as white noise. The random walk error model shows relations in (2.4.2).

$$\begin{aligned} \varepsilon_k &= \varepsilon_{k-1} + w_{k-1} \\ \sigma_k^2 &\stackrel{def}{=} \langle \varepsilon_k^2 \rangle = \sigma_{k-1}^2 + \langle w_{k-1}^2 \rangle = \sigma_0^2 + k \cdot Q_w \text{ for static systems} \\ Q_w &\stackrel{def}{=} E_k \langle w_k^2 \rangle. \end{aligned} \tag{2.4.2}$$

The Q_w value units are squared error per time step Δt . For example, gyro errors might be specified in deg/\sqrt{h} and most navigation-grade gyros have this errors in order of $10^{-3} deg/\sqrt{h}$ or less.

Harmonic noise

Temperature control systems introduce often-cyclical errors because of thermal transport lags. These can cause harmonic errors in sensor outputs, with periods scaled with device dimensions. In addition, ambient devices may be a source of other harmonic noise and that can excite acceleration-sensitive error sources in sensors we use.

1/f noise

This error source is characterized by power spectral density that falls off with factor $1/f$, where f is the frequency. It is present in all electronic devices and it is usually modelled as a combination of white noise and random walk errors.

Fixed-pattern errors

Those errors are identified in the sensor output arising from the input-output relationship. If this relationship is known they can be eliminated. There are dead-zone errors and quantization errors, and cumulative effect of both of them is affected by zero-mean input noise or dithering in addition. Cumulative quantization errors for sensors with frequency outputs are bounded by \pm one-half of the least significant bit (LSB) of the digitized output.

Some of more common types of input-output errors are shown in Figure 2.5:

- bias is any nonzero output of sensor when the input equals zero;
- scale factor error is usually caused by manufacturing tolerances;
- nonlinearity is present almost in all sensors, typically up to some degree;

- asymmetry is caused by mismatched push-pull amplifiers and in fact it is scale factor sign asymmetry;
- dead-zone caused by mechanical static friction or lock-in;
- quantization error is inherited in all digitized systems; the mean value may not be zero when constant value is on the input, although it could be noted under calibration conditions.

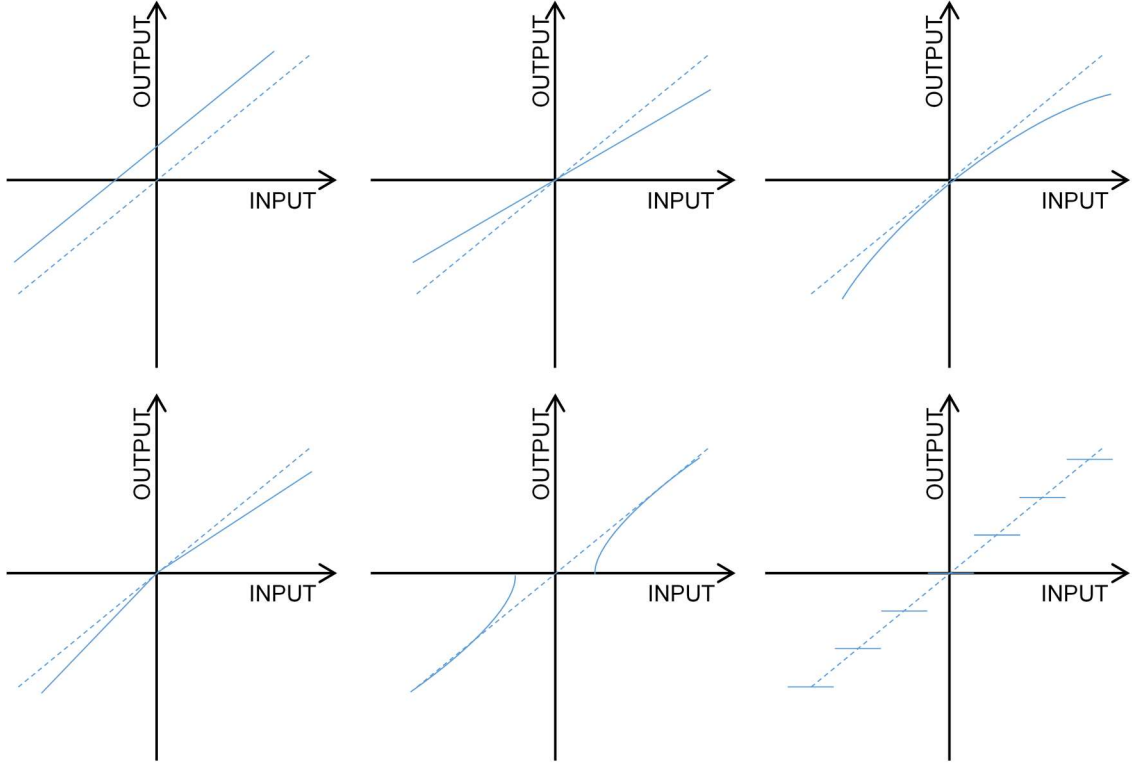


Figure 2.5 Common input-output recoverable errors, in order first line: bias, scale factor, nonlinearity; in order second line: \pm asymmetry, dead-zone, quantization, [13].

2.5 Sensor calibration

To calibrate and compensate offsets, biases, scale factors and misalignments, affine (linear plus offset) model is used. Biases are included in offsets and the rest is linear. When we define output as shown in relation (2.5.1), where \mathbf{z}_{input} is the vector representing the inputs (accelerations or rotation rates), \mathbf{z}_{output} is the vector representing the corresponding outputs, \mathbf{b}_z is the vector of sensor output biases and \mathbf{M} represents the linear input-output model.

$$\mathbf{z}_{output} = \mathbf{M} \cdot (\mathbf{z}_{input} + \mathbf{b}_z) \quad (2.5.1)$$

$$\mathbf{z}_{input} = \mathbf{M}^{-1} \cdot \mathbf{z}_{output} - \mathbf{b}_z \quad (2.5.2)$$

To estimate the values of \mathbf{M} and \mathbf{b}_z , several pairs of given input-output vectors $[\mathbf{z}_{input, k}, \mathbf{z}_{output, k}]$ have to be defined, (2.5.2). These outputs are measured while controlled calibration conditions, thus we get a pair of input-output recorded under these conditions and applicable for sensor compensation. This result can be generalized for a cluster of $N \geq 3$ gyroscopes or accelerometers. For more information, see [13] and [14].

2.6 Orientation determination

The orientation of the inertial measurement unit or its tilt is unknown in a real terrain and it is perhaps the most important step to estimate this state as accurate as possible. Any inaccuracy leads to wrong de-rotation from RPY coordinates to other, inertial coordinates, e.g. NED, ENU or ECEF, [13].

2.6.1 Euler angles

This way, the orientation might be defined as rotation angles about each of axes (vehicle roll, pitch and yaw axis), called Euler angles, named for the Swiss mathematician Leonard Euler (1707-1783). With this approach, it is always necessary to specify the order of rotations when specifying Euler angles.

The rotation from RPY coordinates to NED coordinates can be composed from three Euler rotation matrices, consecutively yaw ψ , pitch θ and roll φ , as is shown in (2.6.1), respectively (2.6.2).

$$C_{NED}^{RPY} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) \end{bmatrix} \quad (2.6.1)$$

$$C_{NED}^{RPY} = \begin{bmatrix} \cos(\psi) \cdot \cos(\theta) & -\sin(\psi) \cdot \cos(\theta) + \cos(\psi) \cdot \sin(\theta) \cdot \sin(\varphi) & \sin(\psi) \cdot \sin(\theta) + \cos(\psi) \cdot \sin(\theta) \cdot \cos(\varphi) \\ \sin(\psi) \cdot \cos(\theta) & \cos(\psi) \cdot \cos(\theta) + \sin(\psi) \cdot \sin(\theta) \cdot \sin(\varphi) & -\cos(\psi) \cdot \sin(\theta) + \sin(\psi) \cdot \sin(\theta) \cdot \cos(\varphi) \\ -\sin(\theta) & \cos(\theta) \cdot \sin(\varphi) & \cos(\theta) \cdot \cos(\varphi) \end{bmatrix} \quad (2.6.2)$$

This approach leads to problem with discontinuity when the pitch angle equals 90 degrees. Roll axis is then pointed upwards and any change in pitch or yaw causes ± 180 degrees changes in heading angle. This is called “gimbal lock” and it is the reason why we do not use Euler angles for the orientation determination of IMUs.

In addition, it depends on the sample rate of angular rate sensing and how precise the sensor is, in the other words, computations of φ , θ and ψ during the time from gyroscope outputs, body angular rates, are mathematically very complicated.

2.6.2 Rotation vector

The other possibility is to use the rotation vectors. If the origins of two right-handed orthogonal coordinate systems are the same points, we can define the transformation between those systems by a single rotation about fixed axis, so we need the direction (rotation axis) and magnitude (rotation angle) of transformation.

This method brings two disadvantages. At first, adding multiples of $\pm 2\pi$ to the rotation angle may cause unwanted changes in transformation it represents, and the other one, rotation angle expression is nonlinear and thus it is complicated to find a relation for more consecutive rotations (the function of all previous rotations) by one rotation vector.

The transformation of the rotation vector to the matrix and the transformation of the matrix to the rotation vector, such as a detailed explanation, you find in APPENDIX A, and in [13].

2.6.3 Direction cosine matrix

The coordinate transformation matrix between two orthogonal coordinate systems is a matrix of direction cosines between the unit axis vectors of those two coordinate systems. For relations, please see APPENDIX A. In this method, it is necessary to define the rotating coordinates ROT (see Figure 2.6).

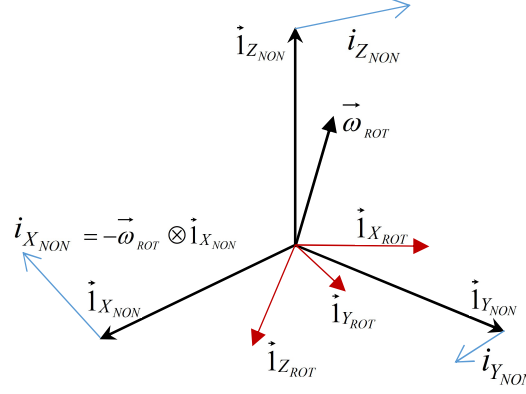


Figure 2.6 Rotating coordinates.

Then, any vector \mathbf{v}_{ROT} in rotating coordinates (ROT) can be expressed in terms of its nonrotating components and unit vectors parallel to the nonrotating axes as (2.6.3), and thus (2.6.4) defines.

$$\begin{aligned} \mathbf{v}_{ROT} &= [v_{xNON} \cdot \vec{\mathbf{1}}_{xNON} + v_{yNON} \cdot \vec{\mathbf{1}}_{yNON} + v_{zNON} \cdot \vec{\mathbf{1}}_{zNON}] \\ &= [\vec{\mathbf{1}}_{xNON} \quad \vec{\mathbf{1}}_{yNON} \quad \vec{\mathbf{1}}_{zNON}] \cdot \begin{bmatrix} v_{xNON} \\ v_{yNON} \\ v_{zNON} \end{bmatrix}, \end{aligned} \quad (2.6.3)$$

$$\mathbf{v}_{ROT} = \mathbf{C}_{ROT}^{NON} \cdot \mathbf{v}_{NON} \quad (2.6.4)$$

where v_{xNON} , v_{yNON} and v_{zNON} are nonrotating components of the vector; $\vec{\mathbf{1}}_{xNON}$, $\vec{\mathbf{1}}_{yNON}$ and $\vec{\mathbf{1}}_{zNON}$ are unit vectors along X_{NON} , Y_{NON} and Z_{NON} axes, as shows Figure 2.6; \mathbf{v}_{ROT} is the vector \mathbf{v} expressed in RPY, analogously \mathbf{v}_{NON} is the vector \mathbf{v} expressed in ECI. Next relation (2.6.5) express the coordinate transformation matrix from nonrotating coordinates to rotating coordinates.

$$\mathbf{C}_{ROT}^{NON} = [\vec{\mathbf{1}}_{xNON} \quad \vec{\mathbf{1}}_{yNON} \quad \vec{\mathbf{1}}_{zNON}] \quad (2.6.5)$$

This transformation is applicable when the static situation is expected, but the gyros measure three nonzero components of the inertial rotation rate vector (2.6.6).

$$\vec{\omega}_{ROT} = \begin{bmatrix} \omega_{xROT} \\ \omega_{yROT} \\ \omega_{zROT} \end{bmatrix} \quad (2.6.6)$$

Finally, we observe a derivative relation of products for time derivation, (2.6.7). This equation was originally used for maintaining vehicle attitude information in strap-down INS implementations.

$$\frac{d}{dt} \mathbf{C}_{ENU}^{RPY} = -\left[\vec{\omega}_{ENU} \otimes \right] \cdot \mathbf{C}_{ENU}^{RPY} + \mathbf{C}_{ENU}^{RPY} \cdot \left[\vec{\omega}_{RPY} \otimes \right] \quad (2.6.7)$$

where $\vec{\omega}_{RPY}$ is the vector of inertial rates measured by the gyro and following applies (2.6.8) - (2.6.11):

$$\vec{\omega}_{ENU} = \vec{\omega}_{earthrate} + \vec{\omega}_{vE} + \vec{\omega}_{vN} \quad (2.6.8)$$

$$\vec{\omega}_{\oplus} = \omega_{\oplus} \cdot \begin{bmatrix} 0 \\ \cos(\phi_{geodetic}) \\ \sin(\phi_{geodetic}) \end{bmatrix} \quad (2.6.9)$$

$$\vec{\omega}_{vE} = \frac{v_E}{r_T + h} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (2.6.10)$$

$$\vec{\omega}_{vN} = \frac{v_N}{r_M + h} \cdot \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} \quad (2.6.11)$$

and

ω_{\oplus} is earth rotation rate;

$\phi_{geodetic}$ is geodetic latitude;

v_E is the east component of velocity with respect to the surface of the earth

r_T is the transverse radius of curvature of ellipsoid

v_N is the north component of velocity with respect to the surface of the earth

r_M is the meridional radius of curvature of ellipsoid

h is altitude above (+) or below (-) the reference ellipsoid surface (\approx mean sea level).

Unfortunately, this equation was finally found to be not well suited for accurate integration in finite-precision arithmetic, so the next, last approach, eventually solves the integration problem, [13].

2.6.4 Quaternions

Quaternions are members of a noncommutative division algebra first invented by William Rowan Hamilton. The idea for quaternions occurred to him while he was walking along the Royal Canal on his way to a meeting of the Irish Academy, and Hamilton was so pleased with his discovery that he scratched the fundamental formula of quaternion algebra (2.6.12), into the stone of the Brougham Bridge, [15].

$$i^2 = j^2 = k^2 = i \cdot j \cdot k = -1 \quad (2.6.12)$$

Quaternions are a single example of a more general class of hyper-complex numbers discovered by Hamilton. While the quaternions are not commutative, they are associative, and they form a group known as the quaternion group [16].

The algebra of quaternions can be defined by using isomorphism between 4x1 quaternion vectors q and real 4x4 quaternion matrices Q , (2.6.13).

$$\mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \leftrightarrow \mathbf{Q} = \begin{bmatrix} q_1 & -q_2 & -q_3 & q_4 \\ q_2 & q_1 & q_4 & q_3 \\ q_3 & q_4 & q_1 & -q_2 \\ q_4 & -q_3 & q_2 & q_1 \end{bmatrix}, \quad (2.6.13)$$

$$= q_1 \cdot Q_1 + q_2 \cdot Q_2 + q_3 \cdot Q_3 + q_4 \cdot Q_4$$

where Q_1, Q_2, Q_3 and Q_4 are quaternion basis matrices, (2.6.14):

$$Q_1 \stackrel{def}{=} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Q_2 \stackrel{def}{=} \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (2.6.14)$$

$$Q_3 \stackrel{def}{=} \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} \quad Q_4 \stackrel{def}{=} \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Quaternion multiplication is noncommutative, the result depends on the order of multiplication. Let's imagine quaternions \mathbf{A} and \mathbf{B} . It applies (2.6.15) and (2.6.16):

$$\mathbf{A} = a_1 \cdot Q_1 + a_2 \cdot Q_2 + a_3 \cdot Q_3 + a_4 \cdot Q_4 \quad (2.6.15)$$

$$\mathbf{B} = b_1 \cdot Q_1 + b_2 \cdot Q_2 + b_3 \cdot Q_3 + b_4 \cdot Q_4$$

and the ordered product \mathbf{AB} is (2.6.16):

$$\mathbf{AB} = (a_1 \cdot b_1 - a_2 \cdot b_2 - a_3 \cdot b_3 - a_4 \cdot b_4) \cdot Q_1 + (a_2 \cdot b_1 + a_1 \cdot b_2 - a_4 \cdot b_3 + a_3 \cdot b_4) \cdot Q_2$$

$$+ (a_3 \cdot b_1 + a_4 \cdot b_2 + a_1 \cdot b_3 - a_2 \cdot b_4) \cdot Q_3 + (a_4 \cdot b_1 - a_3 \cdot b_2 + a_2 \cdot b_3 + a_1 \cdot b_4) \cdot Q_4 \quad (2.6.16)$$

A single quaternion product, the final rotation, is determined by the quaternion product $q_n \times q_{n-1} \dots q_3 \times q_2 \times q_1$, can implement each successive rotation. The quaternion equivalent of the rotation vector $\vec{\rho}$ with $|\vec{\rho}| = \theta$, and where \mathbf{u} is a unit vector, equals then (2.6.17).

$$\mathbf{q}(\vec{\rho}) \stackrel{def}{=} \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \frac{\rho_1}{\theta} \cdot \sin\left(\frac{\theta}{2}\right) \\ \frac{\rho_2}{\theta} \cdot \sin\left(\frac{\theta}{2}\right) \\ \frac{\rho_3}{\theta} \cdot \sin\left(\frac{\theta}{2}\right) \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ u_1 \cdot \sin\left(\frac{\theta}{2}\right) \\ u_2 \cdot \sin\left(\frac{\theta}{2}\right) \\ u_3 \cdot \sin\left(\frac{\theta}{2}\right) \end{bmatrix} \quad (2.6.17)$$

When the two coordinate systems are aligned, the initial value of $\mathbf{q}_{[0]}$ equals $[1 \ 0 \ 0 \ 0]^T$. In inertial measuring systems the initial $\mathbf{q}_{[0]}$ is determined during INS alignment procedure. We can then define the calibrated value of the orientation, quaternion \mathbf{q}_k , as a quaternion product as shows (2.6.18), where \mathbf{q}_{k-1} is a prior value of orientation (a quaternion that is determined from the vector as $[0; v_1; v_2; v_3]$) and $\Delta\mathbf{q}$ is the change in attitude, all represented in quaternion form.

$$q_k = \Delta q_k \times q_{k-1} \times \Delta q_k^* \tag{2.6.18}$$

The attitude representations and rotation sequences for quaternion expressions are available in [17] and [18] for example.

2.7 Artificial neural network

An artificial neural network (ANN) enables to decide how the results of the issue should be, without any equations, relations between physical quantities, and probabilistic filters. It is based on an artificial intelligence (AI), which is the intelligence exhibited by machines or software and such problematics including learning, reasoning, knowledge, planning, communication, perception and the ability to move and manipulate objects, [19], [20] and [21]. It depends on the type and extensiveness of the task that is solved by the ANN. Then the structure and connections inside of ANN are defined and biases and weights of trained network decide about final results. In general, the ANN may look as follows (Figure 2.8) and artificial neuron model is shown in Figure 2.7.

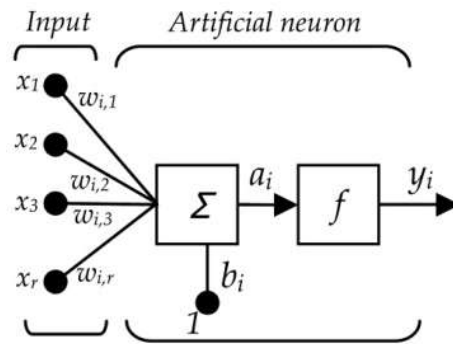


Figure 2.7 The artificial neuron model, [18].

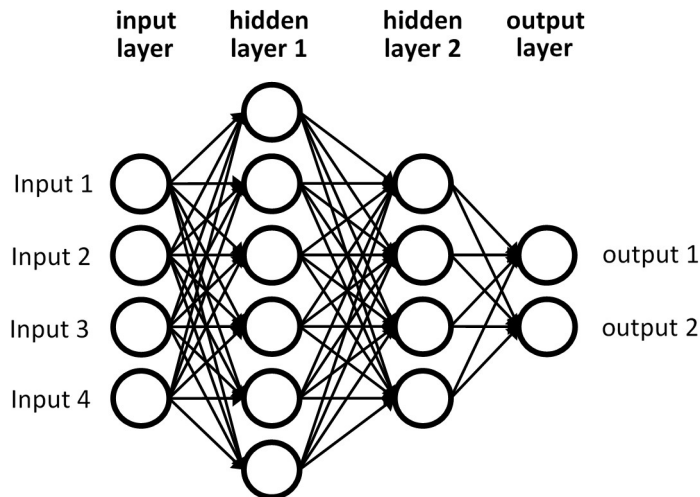


Figure 2.8 The structure of ANN, [22].

Each ANN has its input values representing variables on which the ANN output depends. As an output we consider only one value determining the state of the proposed system. In other cases, more outputs may be present, see [22]. The number of hidden

layers and the number of neurons in particular layers depends on the complexity of the problem we solve, [20], [23].

ANNs were used in systems for tracking, positioning or navigation as it is presented for example in [24], [25] or [26]. Nevertheless, these applications do not use the ANN to find out the state of the system and also these developments combine the IMU with an additional data sources.

The artificial neural networks are used also to solving of many specific types of issues. Always the proper type of the ANN and the method of training and other parameters have to be selected. Here are some kinds of issues:

- Input-output and curve fitting
- Pattern recognition and classification
- Clustering
- Dynamic time series

Our task is to correctly define the state in time. The classification ANN represents an appropriate network for this type of data processing. However, the problem is complex and it is necessary to analyse the data considered in proper time window. Thus we assume that the more complex, dynamic time series type of problem must be solved, [27]. There are lot of kinds and types of ANNs that solve completely different issues, detailed information are provided in Chapter 5.

2.8 Kalman filtering

Kalman filter (KF) (see literature [28] and [29]), also known as linear quadratic estimation (LQE), had become the important instrument for systems that integrate more data sources to give the final solution. You can imagine this filter as an algorithm that uses sets of measurements observed over time (containing random variations of noise) and produces the estimates of unknown variables in order to obtain more precise results (see Figure 2.9). An introduction to concepts gives P. S. Maybeck in [30].

2.8.1 Implementation

With respect to the data from sensors and all other available information, the KF estimates a behaviour by using a form of a feedback control loop. The filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. This procedure is continually repeated to produce current results (see Figure 2.10).

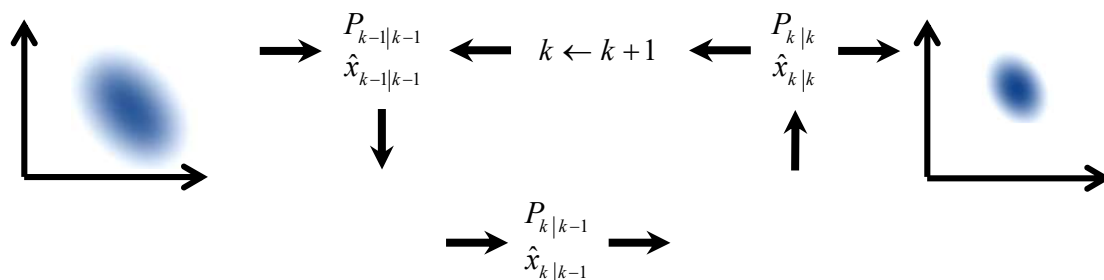


Figure 2.9 Basic concept of Kalman filtering, [35].

For example, in the case of land vehicles positioning, KF gets the data from GNSS unit (GPS, GLONASS and Galileo, Compass or Beidou system) and from inertial unit

(accelerometer, magnetometer and gyroscope), [31]. The adaptive Kalman filtering for low-cost INS/GPS is shown in [32], [33].

It can also get data from systems available in vehicle (data information from ABS and ASR unit, information about steering wheel deflection and data from odometer), as shown in [34]. The filter is very powerful because it supports the estimations of past, present, and even future states, and it can do so even when the precise nature of the modelled system is unknown, [35]. The theory of the optimal state estimation is described in [24].


<p>Correction step:</p> <p>a) Calculate the Kalman Gain</p> $S = HP_k^- H^T + R$ $K_k = \frac{P_k^- H^T}{S}$ <p>b) Correct the a priori state estimate</p> $x_k = x_k^- + K_k (z_k - h(x_k^-, 0))$ <p>c) Correct the a posteriori error covariance matrix estimate</p> $P_k = P_k^- - K_k HP_k^-$	<p>x_k^- the predicted or a priori value of the estimated state vector</p> <p>x_k the corrected or a posteriori value of the estimated state vector</p> <p>z the measurement or observation vector</p> <p>R the sensor noise covariance or measurement uncertainty</p> <p>Q_k the dynamic disturbance noise covariance</p> <p>P_k^- the predicted or a priori value of estimation covariance</p> <p>P_k the corrected or a posteriori value of estimation covariance</p> <p>K Kalman gain</p> <p>A Jacobian of the system model with respect to state</p> <p>H the measurement sensitivity matrix or observation matrix</p> <p>$h(x_k^-, 0)$ the predicted measurement</p> <p>$z_k - h(x_k^-, 0)$ innovations vector</p>
	
<p>Prediction step:</p> <p>a) Predict the state</p> $x_k^- = Ax_{k-1}$ <p>b) Predict the error covariance matrix</p> $P_k^- = A_k P_{k-1} A_k^T + Q_{k-1}$	

Figure 2.10 Kalman filter process, [35].

2.9 Trajectory reconstruction

Trajectory reconstruction is difficult process when the high precision is supposed to be reached and when there is not any support of additional external information system or auxiliary system implemented, [36], [37]. The successive computation of position is called strapdown navigation (Figure 2.11). In addition, heading from the magnetometer should be taken into account. Nevertheless, surrounding environment may differ with the time and place where the measurement is performed. Because of that, the data from magnetometer is not always included into the strapdown IMU system. This issue is discussed in [35].

The essential processing function includes double integration of acceleration to obtain the position. The measured angular rates are also integrated to maintain the knowledge of the IMU orientation. The initial position, velocity and orientation must be known before the initialization of integration [13]. The long-term evaluation of the orientation, velocity and position in time brings high inaccuracies into this results. In

addition, the inaccuracy in determination of the orientation causes additive de-rotation of the measured acceleration in IMU's body frame and it induces incorrect subtraction of the Earth gravitational force. Figure 2.11 shows the simple strapdown INS and its outputs.

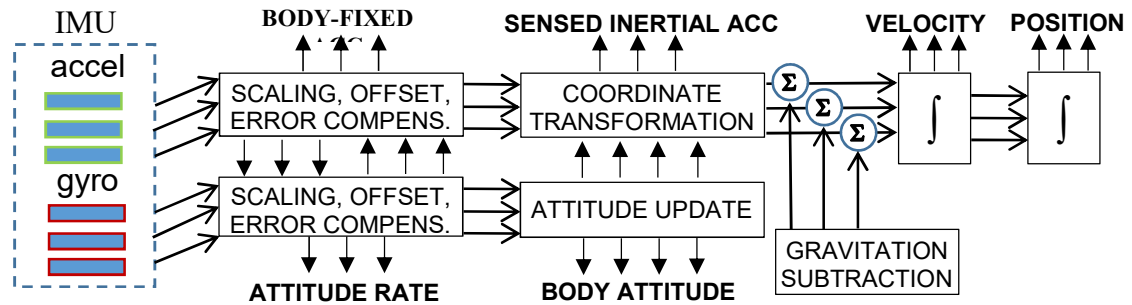


Figure 2.11 Simple strapdown INS and its outputs [13].

2.10 Problems

- During the measurement, drifts and offsets arise on the output of the calibrated gyroscopes. The IMU orientation is defined (RPY to ENU). With time, the inaccuracy of orientation determination rises and thus the velocity and position is computed with enormous errors (in direction and in size).
- The acceleration also drifts during the time. Then the measured acceleration is not exactly 1 g while the IMU stays still. When the IMU stays still, the acceleration may be averaged and normalised. Nevertheless, during the walk or any other motion, the acceleration drift is not fully compensated and thus the estimated velocity (and position) may differ from the true values due to the integration of the acceleration.
- Metal objects placed close to the IMU affects the magnetometer output. It is called as soft iron offset. We assume that when the metal objects are present at a distance of at least fifteen centimetres, they do not affect measured values with the impact on the result (the magnetic north determination). The hard iron offset (the effect of the PCB, electronical components, etc.) has to be suppressed.

3 DISSERTATION OBJECTIVES

In Chapter 2, the recent developments in the field of inertial navigation systems and inertial positioning systems were presented. The analysis clearly identified new directions of the future research:

- SYSTEM FOR TERRESTRIAL EVALUATION OF THE CURRENT STATE

This may furnish for example GNSS navigation when the signal is lost, but, also and above all, this may be used for terrestrial indoor navigation or position determination for short distances, up to several meters, while walking, jogging, driving, etc.

The task is to develop a system that works without any step detection algorithms and map assigning, purely based on sensor outputs processing, with sufficient accuracy. It is also desirable to get a system that can be hold in the hand during its operation.

Therefore, we can define the following objectives of the dissertation:

- To develop the method for determination of the sensors orientation with respect to the navigation coordinates using only the sensor outputs while the system is essentially stationary.
- To develop the method for determination of the sensors orientation with respect to the navigation coordinates using only the sensor outputs while the system is not stationary and while it moves.

Those tasks lead to coordinate alignment ability and thus to ability of subtraction of split g-force (measured gravitational acceleration) from particular axes with eminent focus on accuracy.

- To create an artificial neural network (ANN) that recognizes and defines “what is going on” with the system and to implement it.

It leads to reduction of the positioning errors due to parasitic sensed rotation rates and accelerations. It also ensures that the integration errors will not be cumulated during whole measurement.

- To create a Kalman filter; that is a necessary element where data from the inertial sensors are used for the position determination.

After performing all these tasks, integrations may follow and the velocity and position in time may be computed as well as the trajectory of the moving object can be reconstructed. In addition, the determination of the unit orientation and heading as a function of time is available.

- To develop an IMU with application that evaluates all previous tasks and presents results.

Developed IMU will be based on proper modules with 3-D accelerometer, 3-D rotation rate sensor and 3-D magnetometer connected to appropriate MCU. Proposed application running on PC will process data from the IMU and MCU including graphical representation of results and verification of the system in which a new method of fully inertial positioning is implemented.

- To perform and evaluate series of experiments.

The complex system will be experimentally tested in different scenarios to verify improvements in positioning.

4 INERTIAL MEASUREMENT UNIT

In this Chapter the hardware, firmware and data acquisition techniques are presented. At the end, calibration and compensation process for used sensors is demonstrated. These steps lead to obtaining of the basic orientation and linear acceleration used for IMU positioning from inertial sensors.

4.1 Hardware

As a suitable equipment we chose 9-DOF sensor module. Firstly, we used a module MinIMU-9 v2 with an accelerometer, a gyroscope and a magnetometer, all of them 3-axial (see Figure 4.1). All sensors use MEMS technology with communication I²C interface. This module was connected to common MCU combining processor ATmega16L and 32 Mbit flash memory. Measured data (3-D acceleration, 3-D rotation rate and 3-D magnetic field in Cartesian coordinates) were stored into the flash memory. After acquisition they were copied to the PC and processed in MATLABTM [38]. Algorithms for positioning have been developed, including artificial neural network and Kalman filters.

This hardware represents a cheap solution, but its low computing power performance caused in very low data rate and insufficient positioning for real scenarios. For example, when the walk motion is estimated, the rate of data sensing should be about 80 Hz. Above mentioned module MinIMU-9 with MCU provided about 16 timestamps per second. The key problem of this solution is complicated access to memory for real-time application (continuous time experiment). Thus, it was necessary to find another solution.

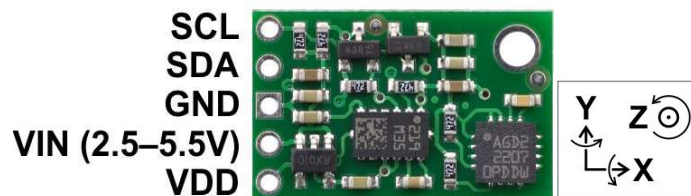


Figure 4.1 MinIMU-9 v2 board and sensor axis orientation, [46].

As a second hardware X-NUCLEO-IKS01A1 board was chosen. It consists of motion MEMS and environmental sensors. It is compatible with Arduino Uno. Measured data are sent by BT or by USB cable to a PC and may be processed in real time or saved for further processing. This sensor board is designed around STMicroelectronics' LSM6DS0 3-axis accelerometer, 3-axis gyroscope, the LIS3MDL 3-axis magnetometer and in addition, the HTS221 humidity and temperature sensor and the LPS25HB* pressure sensor is available.

Information and technical parameters are in APPENDIX C.

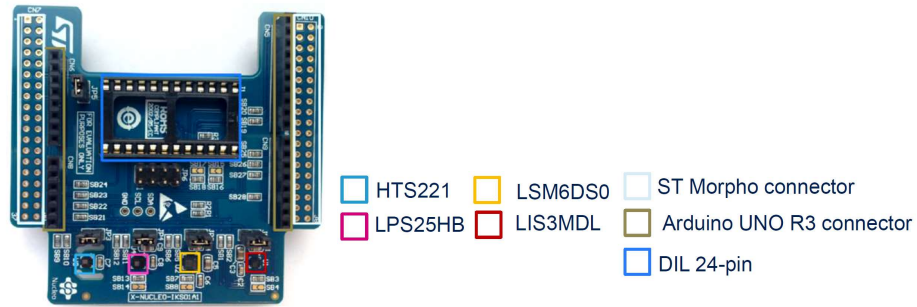


Figure 4.2 X-NUCLEO-IKS01A1 board [46].

4.2 Firmware

The function of Arduino UNO is to read data from sensors and send them directly to the PC. I²C protocol is used for data retrieval from the sensors. Arduino Uno was programmed in Arduino IDE application using a programming language C++.

The firmware code is divided into two sections, the first part (setup function) is performed on start-up of Arduino and applies the settings, as shows the example part of code below. Further information about the register settings are listed in APPENDIX C, and APPENDIX D. Arduino is set as a master and sensors are set as slaves.

```

void setup()
{
  Wire.begin();           // I2C init
  Serial.begin(115200);   // SP init

  setupMag();             // Magnetometer registers setting
  setupAG();              // Accelerometer and gyroscope
                          // registers setting
}
    
```

The second part (loop function) reads the sensor data and sends them to the PC, as it is shown in next code part. Whole communication cycle captured by oscilloscope is shown in Figure 4.3. The magenta curve represents Arduino TX in time, yellow curve represents I²C SDA and cyan curve represents I²C SCL. For more details on communication, please see APPENDIX D.

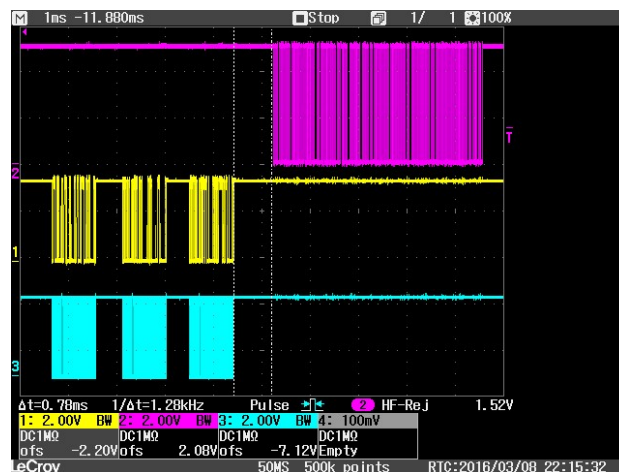


Figure 4.3 One Arduino communication cycle, consequently. TX, SDA, SCL.

```

void loop()
{
  String output = "";
  readFrom(30, B00101000, 6);           // Magnetometer reading

  while (Wire.available())
  {
    short c = Wire.read();
    c += (Wire.read() << 8);
    output += String(c, DEC) + " ";
  }

  readFrom(107, B00011000, 6);         // Gyroscope reading

  while (Wire.available())
  {
    short c = Wire.read();
    c += (Wire.read() << 8);
    output += String(c, DEC) + " ";
  }

  readFrom(107, B00101000, 6);         // Accelerometer reading

  while (Wire.available())
  {
    short c = Wire.read();
    c += (Wire.read() << 8);
    output += String(c, DEC) + " ";
  }

  output += String(millis(), DEC) + " "; // timestamp

  if(digitalRead(buttonPin) == HIGH)   // button state
    output += "0";
  else output += "1";

  Serial.println(output);               // send to PC
  delay(7);
}

```

4.3 Data acquisition

The firmware in Arduino defines the format in which the data are sent. In the source code, the package contains measured accelerations in x , y and z -axis from accelerometer (Figure 4.5), angular rate in x , y and z -axis from gyroscope (Figure 4.4) and magnetic field in x , y and z -axis from magnetometer (Figure 4.6). There is the possibility to receive the time stamp to get precise Δt , in other words, to get accurate time between two samples. In addition, Arduino sends the button state. This button allows to receive additional boolean value that is defined by user.

Axes of all sensors have essentially the same origin but magnetometer has a different right-handed axis system than accelerometer and gyroscope. Thus, we have to rotate measured magnetic field vector by $+90$ degrees along common z -axis and then transform the coordinate system of magnetometer using transformation matrix (2.3.1) to get one united vehicle-fixed coordinate system RPY (Figure 4.7).

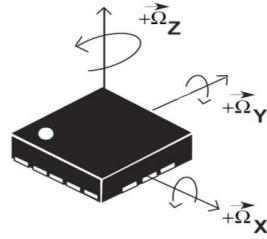


Figure 4.4 Direction of detectable angular rates.

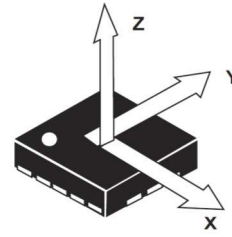


Figure 4.5 Direction of detectable accelerations.

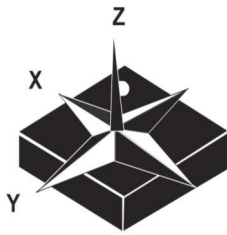


Figure 4.6 Direction of detectable magnetic field.

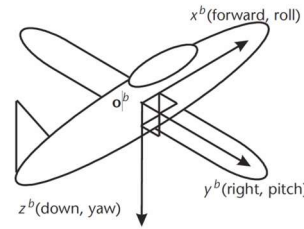


Figure 4.7 Body-fixed RPY (roll-pitch-yaw) axis.

Arduino sends just a raw data from converters.

Based on the register settings, the acceleration is measured in range of ± 2 g and the sensitivity is then 0.061 mg/LSB. The rotation rate sensor is set to range of ± 500 dps, the sensitivity is then 17.5 mdps/LSB. The magnetometer measures magnetic field in range of ± 4 gauss and the sensitivity is then 0.146 mGauss/LSB. Resolution of all sensors is 16 bits.

Typical magnetic field at places where the measurements were performed is about 48.897 nT (488.977 mGauss), see TABLE 4.1.

TABLE 4.1 Magnetic field values in Kohoutovice, Brno [45].

Model Used: WMM2015							
Latitude: 49.1962214° N							
Longitude: 16.5407075° E							
Elevation: 360.0 m GPS							
Date	Declination (+ E - W)	Inclination (+ D - U)	Horizontal Intensity	North Comp (+ N - S)	East Comp (+ E - W)	Vertical Comp (+ D - U)	Total Field
2015-04-16	4.1030°	65.4184°	20340.9 nT	20288.7 nT	1455.4 nT	44466.1 nT	48897.7 nT
Change /year	0.1218°	0.0068°	7.4 nT	4.3 nT	43.7 nT	30.2 nT	30.5 nT
Uncertainty	0.36°	0.22°	133 nT	138 nT	89 nT	165 nT	152 nT

Finally, we get 11 values from 9-DOF device per sample (this is what the device sends: [$mag_x, mag_y, mag_z, gyr_x, gyr_y, gyr_z, acc_x, acc_y, acc_z, time, button$]), and those sets are sent with frequency of about 86 Hz.

The required sampling frequency is given by the spectrum of the measured data during typical and particular movements. For the walk and staying still the sampling frequency about 80 Hz is satisfactory. For the data processing while the IMU motion is

running or flying, higher sampling frequency is required. For graphic representation see amplitude spectrums below: Figure 4.8 - Single-sided amplitude spectrum of the sensor's data – walk, Figure 4.9 - Single-sided amplitude spectrum of the sensor's data – swinging.

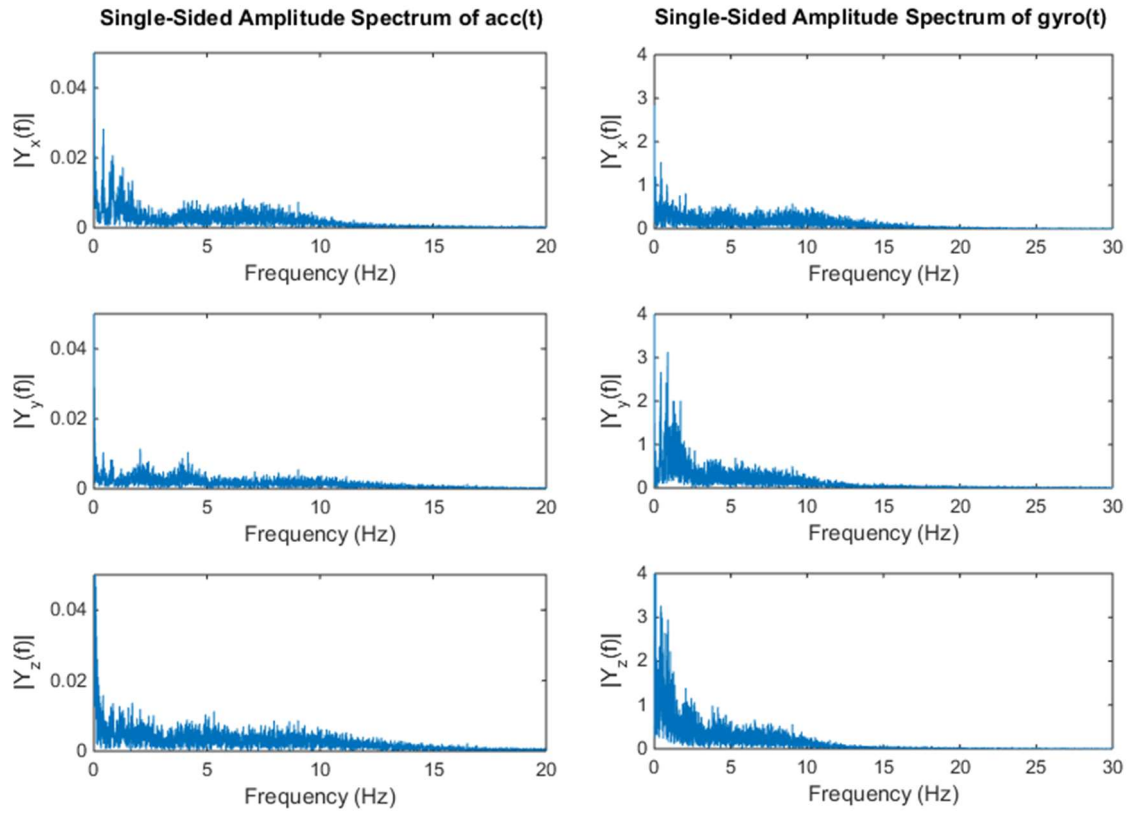


Figure 4.8 Single-sided amplitude spectrum of the sensor's data – walk.

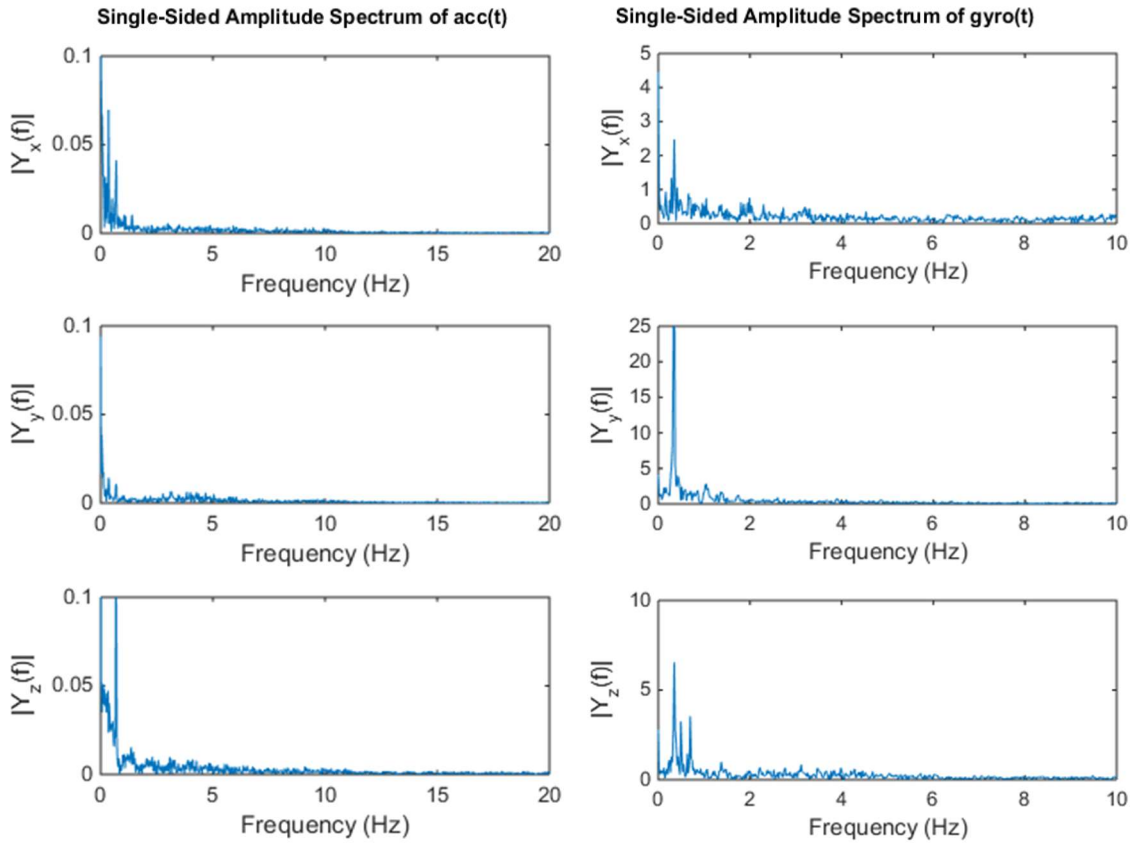


Figure 4.9 Single-sided amplitude spectrum of the sensor's data – swinging.

4.4 Sensor calibration and compensation

Sensor calibration is the process determining the parameters of the compensation model. Sensor compensation is the process recovering the sensor inputs from the sensor outputs.

Figure 4.10 shows the rotation rate measured by the rotation rate sensor when the IMU was laying on the table for 30 minutes without any movement. The next figure (Figure 4.11) shows the integration of the measured data in degrees in time. The detailed figure shows first ten seconds of the measurement. You can see that the data are almost correct for the first three seconds, then the error rises significantly.

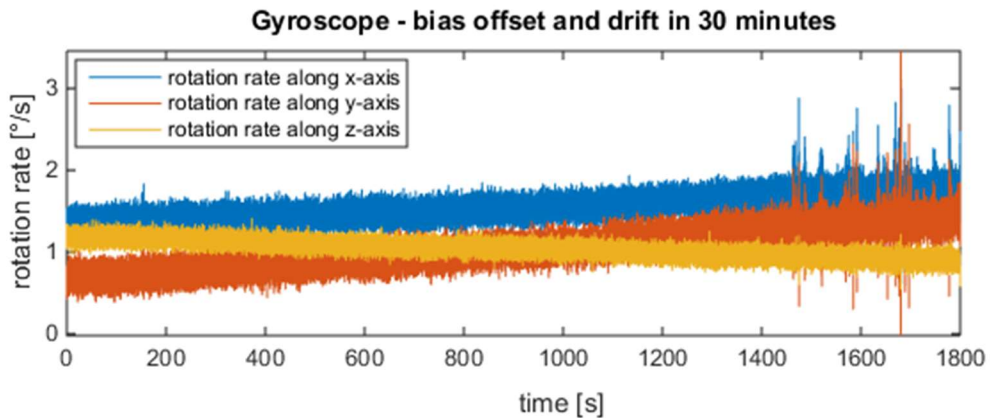


Figure 4.10 Raw rotation rate in time when IMU is still – 30 minutes.

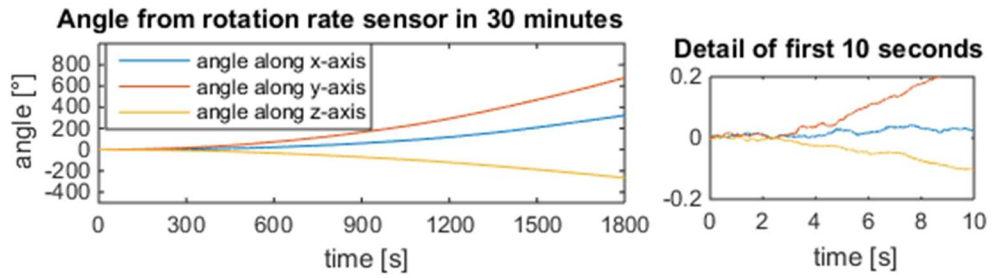


Figure 4.11 The error in angle determination caused by offset and drift.

The integration of the biases of the accelerometer (see Figure 4.12) cause the error in the position determination, this error increases quadratically over time. The derotation have also more significant error due to gyroscope biases integration. The bias of the gyroscope accumulates the position error over time proportional to the cubic function of the time index. The peaks at the end of the measured data are given by typing on a keyboard while the IMU was lying on the same table as the keyboard 1 meter away.

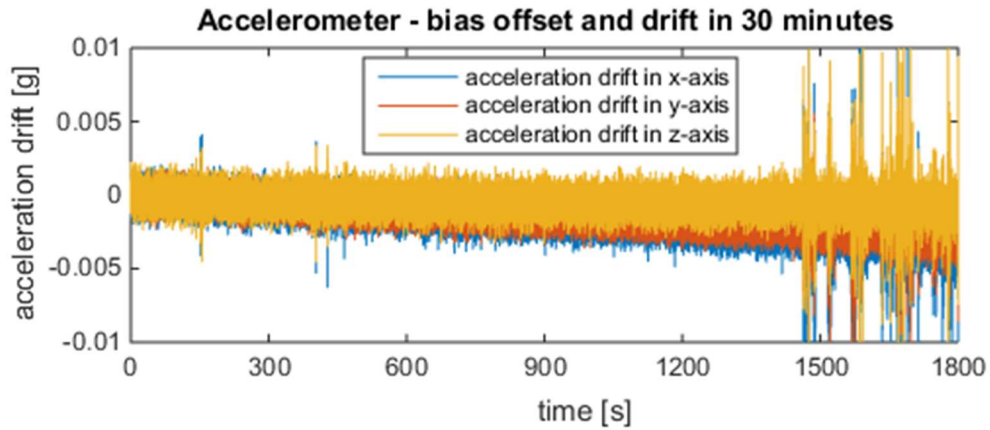


Figure 4.12 Raw acceleration after the expected value subtraction when the IMU is still

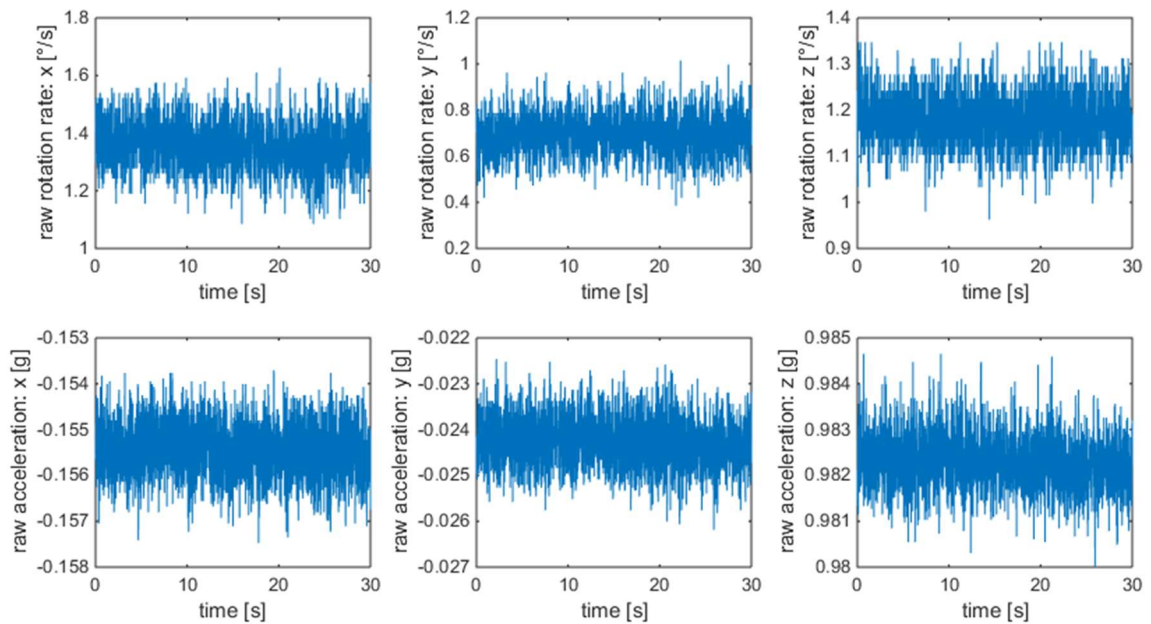


Figure 4.13 Raw rotation rates (top) and raw accelerations (bottom) in particular axis.

In Figure 4.13, the raw rotation rates and raw accelerations in particular axes (in order x-axis, y-axis and z-axis from left to right) are shown – the 2520 samples are taken over

a period of 30 seconds. The next Chapters show the biases determination for both of the sensors.

4.4.1 Bias of the accelerometer

The vector of the average output in the body frame of accelerometer is defined by (4.4.1).

$$\hat{F}_b = [\hat{f}_x, \hat{f}_y, \hat{f}_z]^T \quad (4.4.1)$$

This is the sum of the correct acceleration vector F_b in the body frame and the bias vector of the accelerometer in the body frame, (4.4.2).

$$\hat{F}_b = F_b + \nabla^b(0) \quad (4.4.2)$$

After the derotation of acceleration vector into inertial coordinate system, we expect that the acceleration vector equals $[0, 0, 1]^T$. The transformation from RPY to ENU coordinates is defined as:

$$C_{ENU}^{RPY} = [\vec{I}_R \quad \vec{I}_P \quad \vec{I}_Y] = \begin{bmatrix} \vec{I}_E^T \\ \vec{I}_N^T \\ \vec{I}_U^T \end{bmatrix} \quad (4.4.3)$$

$$= \begin{bmatrix} S_Y C_P & C_R C_Y + S_R S_Y S_P & -S_R C_Y + C_R S_Y S_P \\ C_Y C_P & -C_R S_Y + S_R C_Y S_P & S_R S_Y + C_R C_Y S_P \\ S_P & -S_R C_P & -C_R C_P \end{bmatrix}$$

where Y represents yaw angle, P represents pitch angle, R represents roll angle,

$$S_R = \sin(R) \quad (4.4.4)$$

$$C_R = \cos(R) \quad (4.4.5)$$

$$S_P = \sin(P) \quad (4.4.6)$$

$$C_P = \cos(P) \quad (4.4.7)$$

$$S_Y = \sin(Y) \quad (4.4.8)$$

$$C_Y = \cos(Y) \quad (4.4.9)$$

For details see APPENDIX A. The bias of the used accelerometer is then defined by (4.4.10). The example data are given by the average of the measured data while the IMU was laid on a table without any movement.

$$\nabla^b(0) = \hat{F}_b - F_b = \begin{bmatrix} -0.1601910 \\ -0.0275174 \\ 0.986702 \end{bmatrix} - \begin{bmatrix} -0.1602160 \\ -0.0278812 \\ 0.986698 \end{bmatrix} = \begin{bmatrix} 2.5e-05 \\ 3.638e-04 \\ 4e-06 \end{bmatrix} \quad (4.4.10)$$

where \hat{F}_b is the average measured acceleration vector and F_b is a vector $[0, 0, 1]^T$ affected by the IMU's orientation (bank, elevation and heading angle – BK , EL , H in

order) found out during the stationary stage. The BK , EL and H value was determined from the quaternion that defines the absolute orientation of the IMU in ENU coordinates (averaged). Those values are:

$$H = 0.128828^\circ, EL = 9.219442^\circ, BK = -1.597682^\circ$$

and F_b is determined by (4.4.11):

$$F_b = \begin{bmatrix} -\sin \beta \cos EL \\ \sin EL \\ \cos BK \cos EL \end{bmatrix} \cdot 1. \quad (4.4.11)$$

where $\beta = \sin^{-1}\left(\frac{\sin BK}{\cos EL}\right)$; the heading H does not appear here. Those angles are not equal to the roll, pitch and yaw angle in RPY coordinates.

4.4.2 Bias of the gyroscope

Theoretically, the biases of the rotation rate sensors are caused by the Earth rotation rate. The real IMU gyroscope measures the sum of the Earth rotation rate Ω_{ie} and the bias of the rotation rate sensor ∇^b as shows the relation (4.4.12). As [45] and [48] presents, the earth rate is approx. $7.292115090 \cdot 10^{-5}$ rad/s ($4.178074184 \cdot 10^{-3}$ °/s). This value may be ignored, the resolution of used IMU is higher (the sensitivity is $15.258789 \cdot 10^{-3}$ °/s when the measuring range of particular gyro axes is ± 500 °/s) and equation (4.4.13) is applied. In order to remove the gyro bias, the average rotation rate vector is subtracted from the measured rotation rate vector.

$$\hat{w}^b = \Omega_{ie} + \nabla^b \quad (4.4.12)$$

$$\nabla^b = \hat{w}^b = \begin{bmatrix} \hat{\omega}_x^b \\ \hat{\omega}_y^b \\ \hat{\omega}_z^b \end{bmatrix} \quad (4.4.13)$$

4.4.3 Calibration and compensation

The calibration is divided into two processes in this case. The first one is called “hard” sensor compensation and calibration and it is performed always when the IMU is switched on. During this “hard” calibration process the biases of the accelerometer and gyroscope are subtracted, the scale factor for the accelerometer data is defined and magnetometer data are adjusted in way that the magnetic field strength is adapted to the location of measurement (the location should be chosen in the application). Geographic coordinates of the selected locations are available in APPENDIX B, part B.1.

The second type of calibration, that may be called as recalibration or “soft” calibration, is performed automatically when the state of the IMU is defined as the “still”. The acceleration vector is scaled – values are normalised. Based on the measured rotation rates the new calibration constants are found out and applied when the IMU state changes to the “walking” as the new bias offsets.

In Figure 4.14 you can see the raw received data from the sensors, converted to appropriate units ([g] for the acceleration; [°/s] for the rotation rate; [mGauss] for the magnetic field magnitude). On the right side of the charts, the last measured values are

shown (in order x -axis, y -axis and z -axis value). This graph shows the one turn in clockwise direction by 360° , when the blue curve represents the x -axis, the green curve represents the y -axis and the red curve represents the z -axis. The x -axis of the graph represents time in [s], as elsewhere throughout the document of the same graph type.

We are not able to determine the heading of the IMU unless we have data from the calibrated magnetometer and we know the magnetic declination (δ) of the place where the measurement was performed. The rotation to the flat position must be applied to the calibrated magnetometer data in order to determine the heading correctly in 3-D. The magnetic declination is an angle in the horizontal plane between magnetic north (where the compass needle points, corresponding to the direction of the Earth's magnetic field lines) and true north (geographic North Pole). See TABLE 4.2 for information about the declination in Kohoutovice, Brno.

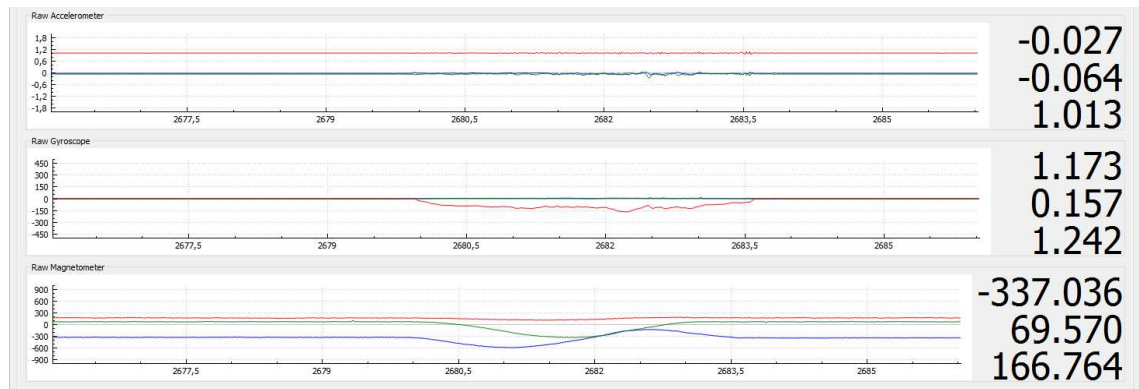


Figure 4.14 Data from uncalibrated sensors in time [s].

TABLE 4.2 Magnetic declination for Kohoutovice, Brno [47].

Model Used:	WMM2015	
Latitude:	49.1962214° N	
Longitude:	16.5407075° E	
Date	Declination	
2015-04-16	4.02° E ± 0.36° changing by 0.12° E per year	

Then, Figure 4.15 shows calibrated magnetometer data and $\sin(x)$ and $\cos(x)$ function that is formed when the IMU rotates along a single axis by 360° with starting and ending heading equals δ – the x -axis points straight to magnetic north.

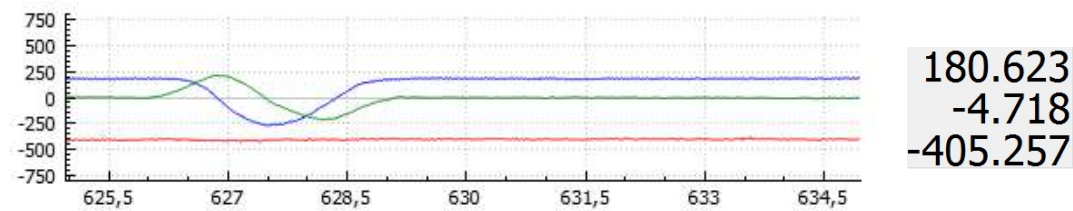


Figure 4.15 Calibrated magnetometer data (360° rotation) [mGauss] in time [s].

A calibration of accelerometer and magnetometer are based on the same principle. Because of static nature of quantities measured by those sensors, we can measure minimum and maximum value in each axis (in any orientation) as the calibration constants.

In the case of magnetometer, the maximum values were reached when particular axes were pointing in the same direction as the vector of magnetic field. In the case of accelerometer, maximum values were reached when particular axes were pointing downwards, in the direction of gravitational acceleration. The scale factor was then computed based on the knowledge of the strength of Earth's gravity and Earth's magnetic field.

The (4.4.14) and (4.4.15) equations have been used to reflect the calibration constants in measured data. The $\bar{\mathbf{a}}$ is an arithmetic average of minimum and maximum value of measured acceleration (magnetic field), separately determined for each axis. The first relation removes sensor offset. The second relation scales measured values to the interval in which the values should be. The acceleration is correct due to the multiplication by 1 g. The constant 489.151 for magnetometer scales the measured magnetic field into the range of the total magnetic field on the specific place on the Earth's surface (see APPENDIX B, part B.3). The part of the code is shown in Figure 4.16. In fact, finally this is expressed in a matrix form.

$$\mathbf{z}_{output} = \mathbf{z}_{input} - \bar{\mathbf{a}}_{\min, \max} \quad (4.4.14)$$

$$\mathbf{z}_{output}^* = \frac{2 \cdot \mathbf{z}_{output}}{\Delta_{\min, \max}} \quad (4.4.15)$$

```
//Offset
acc.setX(acc.x() - ((accMax.x() + accMin.x()) / 2));
acc.setY(acc.y() - ((accMax.y() + accMin.y()) / 2));
acc.setZ(acc.z() - ((accMax.z() + accMin.z()) / 2));

mag.setX(mag.x() - ((magMax.x() + magMin.x()) / 2));
mag.setY(mag.y() - ((magMax.y() + magMin.y()) / 2));
mag.setZ(mag.z() - ((magMax.z() + magMin.z()) / 2));

//Scale
acc.setX(1*acc.x() / ((accMax.x() - accMin.x()) / 2));
acc.setY(1*acc.y() / ((accMax.y() - accMin.y()) / 2));
acc.setZ(1*acc.z() / ((accMax.z() - accMin.z()) / 2));

mag.setX(489.151*mag.x() / ((magMax.x() - magMin.x()) / 2));
mag.setY(489.151*mag.y() / ((magMax.y() - magMin.y()) / 2));
mag.setZ(489.151*mag.z() / ((magMax.z() - magMin.z()) / 2));
```

Figure 4.16 C++ source code – a part of calibration [Qi].

After the calibration when the IMU is still, the magnetometer shows magnetic field with total strength of 489.151 mGauss, the accelerometer shows the total gravitational acceleration of +1 g. The gyroscope is calibrated every time when the unit is switched on and drifts are subtracted to achieve zero rotation rate when the IMU does not rotate (and does not move). During the measurement when the IMU is still, the calibration constants are adjusted accordingly.

Recalibration may be done also manually during the measurement. Nevertheless it is necessary to ensure the conditions for the calibration. The measurement of the still-shake-still state of the IMU is shown in Figure 4.17.

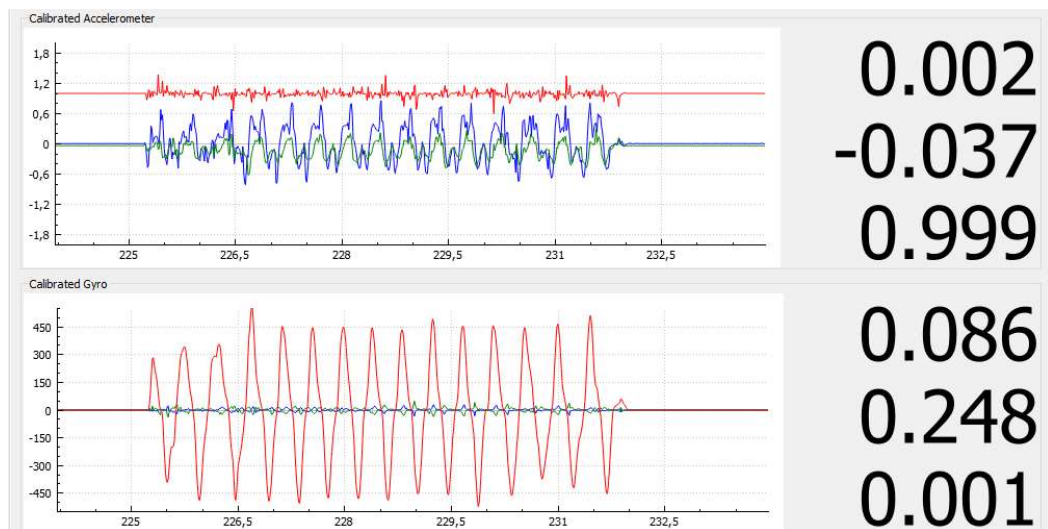


Figure 4.17 Data from calibrated sensors while shaking.

As you can see, the rotation rate and the acceleration show “what is going on“ with the IMU in the body coordinate frame in time. The IMU is placed on the table but it is not perfectly in flat position (in a horizontal plane). The y -axis of the gyroscope has the value of $0.248 \text{ }^\circ/\text{s}$ after the movement. This is the drift and it has to be suppressed for further processing. Thus, the calibration of the gyroscope is necessary during the measurement. This is exactly what the “soft” calibration performs.

5 NEURAL NETWORKS

The new proposed approach is primarily based on the artificial neural network that is designed in order to determine state – “what it is going on”. In this work, recognition of two states are presented. The first case is that the IMU walks, the second case is that the IMU is static regardless of its orientation. In principal, further states may be added, for example jogging, running, driving, riding, shaking, flying, falling etc.

The very important piece of information is that when the ANN determines the state of the IMU incorrectly, there are two cases of the wrong decision.

1. The ANN determines the walk and the IMU is still
2. The ANN determines that the IMU stays still and the IMU walks.

The first case does not bring complications, however it is undesirable, since the ANN does not improve the positioning. The second case is unacceptable, the orientation of the moving IMU is recalculated as it is “still” (from the actual accelerometer data). The further processing of data after the incorrect determination of the orientation causes the error with a very high severity.

5.1 Principle of ANN

I decided to create time-delayed neural network, since the static values of one sample do not have any predictive value. Figure 5.1 shows principle of convenient ANN. It is nonlinear autoregressive network with external input (NARX) [41]. More simple type of network and also very convenient is FFTD (feed-forward time-delayed) network, [42], this type does not carry older outputs into the input for to get new output (the principal scheme is in Figure 5.2).

To obtain the training data we recorded a walk with stops. The person holding the IMU used the button to determine if he is walking or is standing still. Then, the neural network was trained with the input set consisting of measured data and the button state as the target output.

A part of example script, code for MATLAB™, is shown in Figure 5.7. This script was written for automatic creation, training and simulation of FFTD artificial neural network. While the train function is given, the time delay (the number of previous samples used) and the number of neurons in hidden layer were sweeping. The training set is divided to three blocks - training part, validation part and test part. The goal criteria have to be set properly.

Once the neural network is trained (one of the given criterion is reached), the ANN structure and its constants (weights, biases and others) are saved with information about time delay and number of neurons in hidden layer. Then, the FFTD ANN is used with unknown data (the data set which was not included in the training set) and its results were depicted in graphical form.

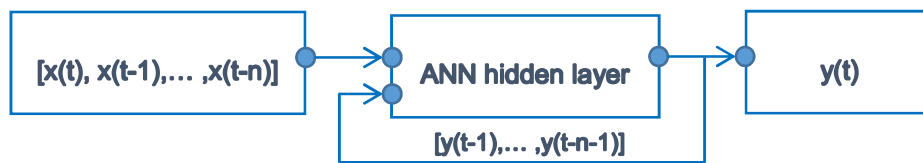


Figure 5.1 NARX ANN principle.



Figure 5.2 FFTD ANN principle.

Parameter and architecture properties such as training function, performance function, divide function, adaptation function, transfer function etc. also have a significant impact on ANN output.

Following text describes the variables and the most suitable functions for the input-output time-series problem with a time-delay neural network.

TRAINING FUNCTION (net.trainFcn)

trainlm Levenberg-Marquardt backpropagation is recommended for most problems, but for some noisy and small problems.

Our training occurs according to trainlm training parameters, shown here (default values are shown, see TABLE 5.1) with their default values:

TABLE 5.1 Default training parameters [MATLAB™].

net.trainParam.epochs	1000	Maximum number of epochs to train
net.trainParam.goal	0	Performance goal
net.trainParam.max_fail	6	Maximum validation failures
net.trainParam.min_grad	$1 \cdot 10^{-7}$	Minimum performance gradient
net.trainParam.mu	0.001	Initial mu
net.trainParam.mu_dec	0.1	mu decrease factor
net.trainParam.mu_inc	10	mu increase factor
net.trainParam.mu_max	$1 \cdot 10^{10}$	Maximum mu
net.trainParam.show	25	Epochs between displays (NaN for no)
net.trainParam.showCommandLine	0	Generate command-line output
net.trainParam.showWindow	1	Show training GUI
net.trainParam.time	inf	Maximum time to train in seconds

The validation vectors are used to stop training if the network performance on the validation vectors fails to improving (number of consequent trials is defined in max_fail).

The test vectors are used as a further check that the network is generated well, but do not affect the training.

trainbr Bayesian regulation backpropagation can take longer but obtain a better solution

trainscg Scaled conjugate gradient backpropagation is recommended for large problems as it uses gradient calculations which are more memory efficient than the Jacobian calculations the other two algorithms use.

INITIALIZATION FUNCTION (net.initFcn)

learngd Gradient descent weight and bias learning function, calculates the weight change dW for a given neuron from the neuron's input P and error E , and the weight (or bias) learning rate LR , according to the gradient descent.

learngh Gradient descent with momentum weight and bias learning function, calculates the weight change dW for a given neuron from the neuron's input P and error E , the weight (or bias) W , learning rate LR , and momentum constant MC , according to gradient descent with momentum.

ADAPTATION LEARNING FUNCTION (net.adaptFcn)

adaptwb Sequential order incremental training w/learning functions, that adapt network with weight and bias learning rules.

PERFORMANCE FUNCTION (net.performFcn)

mse Mean squared normalized error performance function. It measures the network's performance according to the mean of squared errors.

- msereg Mean squared error with regularization performance function. It measures network performance as the weight sum of two factors: the mean squared error and the mean squared weights and biases.
- sse Sum squared error performance function. It measures performance according to the sum of squared errors.

TRANSFER FUNCTION (net.transferFcn)

purelin Linear transfer function is used in final layer of multilayer networks.

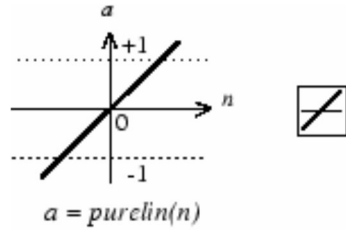


Figure 5.3 Linear Transfer Function.

logsig Log-sigmoid transfer function is commonly used in the hidden layers $\text{logsig}(n) = 1 / (1 + e^{-n})$.

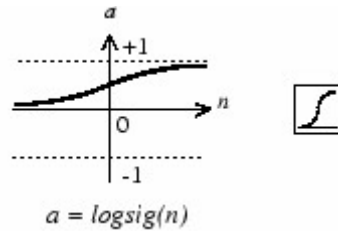


Figure 5.4 Log-sigmoid Transfer Function.

tansig Hyperbolic tangent sigmoid transfer function is used in our neural network in hidden layer; $\text{tansig}(n) = 2/(1+e^{-2*n}) - 1$, mathematically equivalent to $\tanh(N)$.

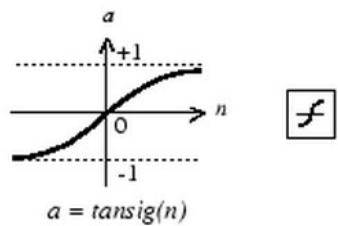


Figure 5.5 Hyperbolic tangent sigmoid Transfer Function.

hardlims Symmetric hard-limit transfer function, $\text{hardlims}(n) = 1$ if $n \geq 0$; -1 otherwise

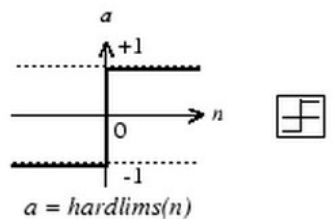


Figure 5.6 Symmetric Hard-Limit Transfer Function.

ARCHITECTURE PROPERTIES

net.numInputs	Number of inputs. My network has 6 x timeDelay input values. Those are [gyr _x gyr _y gyr _z acc _x acc _y acc _z] and their history.
net.numLayers	Number of layers. My network has two layers.
net.biasConnect	Boolean vector, if net.biasConnect(i)=1 (my network), layer i has a bias, and net.biases{i} is a structure describing that bias.
net.inputConnect	numLayer-by-numInputs Boolean vector, when net.inputConnect(i,j)=1, layer i has a weight coming from input j, and net.inputWeights{i,j} is a structure describing that weight.
net.layerConnect	numLayer-by-numLayers Boolean vector, when net.layerConnect(i,j)=1, layer i has a weight coming from layer j, and net.layerWeights{i,j} is a structure describing that weight.
net.outputConnect	1-by-numLayers Boolean vector, if net.outputConnect(i)=1, then the network has an output from layer i, and net.outputs{i} is a structure describing that output.
net.numOutputs	Number of network outputs according to net.outputConnect. My network has one output value. It is [what it is going on].
net.numInputDelays	Maximum input delay according to all net.inputWeight{i,j} delays.
net.numLayerDelays	Maximum layer delay according to all net.layerWeight{i,j} delays.

Further text defines how the time-delay neural network with 15 hidden neurons in one hidden layer and one hidden neuron in output layer may look. Transfer functions are set to tansig in layer 1 and purelin in layer 2.

```

net = network;
net.numInputs = 6;
net.numLayers = 2;
net.biasConnect(1) = 1;
net.biasConnect(2) = 1;
net.inputConnect(1,1:6) = 1;
net.layerConnect(1,1:15) = 1;
net.layerConnect(1,1) = 1;
net.outputConnect(2) = 1;
net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'purelin';

```

This setting is performed by “timedelaynet” script, as shows Figure 5.7. This script was created to finding the best structure of artificial neural network to solving of the time-series problem. The best result was achieved by the neural network that has one

hidden layer with 15 neurons and one neuron in the output layer, with the transfer functions tansig and purelin in order and with the time delay set to 40 samples; that corresponds to approx. 0.5 s of the acquisition.

```

% dly      -timedelay;
% NNUM    -number of neurons in hidden layer;
% X       -INPUT SET;           % W       -measured data from sensors;
% T       -TARGET SET;         % Q       -manual mark from switch;

function [net, Xs, Xi, Ai, Ts, dly, nnum] = TD_TRAINER (DLY, NNUM, W, Q)

X = tonndata(W,true,false);    %preprocess of variables for ANN
T = tonndata(Q,true,false);

for dly=1:1:39
    for nnum=3:1:20
        clear td_net;
        net = timedelaynet(0:dly,nnum, 'trainlm');
        net.input.processFcns = {'removeconstantrows','mapminmax'};
        net.output.processFcns = {'removeconstantrows','mapminmax'};
        net.trainParam.min_grad = 1e-6;
        net.trainParam.goal = 1e-6;
        net.trainParam.max_fail=10;
        net.divideFcn = 'dividerand';
        % net.divideFcn = 'divideblock';
        net.divideParam.trainRatio = 70/100;
        net.divideParam.valRatio = 15/100;
        net.divideParam.testRatio = 15/100;
        [Xs, Xi, Ai, Ts] = preparets(net, X, T);
        net.layers{1}.transferFcn = 'tansig';
        net.layers{2}.transferFcn = 'purelin';
        net = train(net, Xs, Ts, Xi, Ai);
        net.name = ['FFTD_' num2str(nnum) '_' num2str(dly)];
        save('NNS_FFTD.mat', 'net', '-append');

    end;
end;

```

Figure 5.7 MATLAB™ code for verifying the suitability of TD ANN.

As the best-input parameters seem to be a vector of raw data from the accelerometer and a vector of raw data from the gyroscope, both in all three axes. The magnetometer data were discarded due to important dependence on the surrounding magnetic strength.

The ANN may be also trained by adjusted data from the accelerometer and gyroscope, e.g. by gravitational vector length as the first input parameter and rotation vector length as the second input parameter. Nevertheless, trained network showed worse result.

The training of the ANN with chosen structure (see Figure 5.8) is relatively computationally complex. It had been trained on PC (16 GB RAM, Intel® CORE™ 4 x i5-4690K CPU @ 3.50GHz, SSD) using MATLAB™.

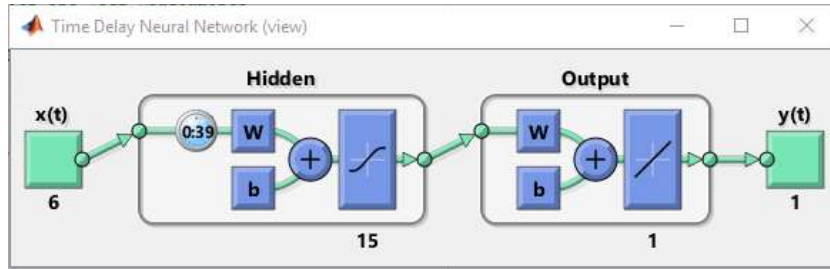


Figure 5.8 ANN structure – feed forward, time delay, purelin transfer function in output layer.

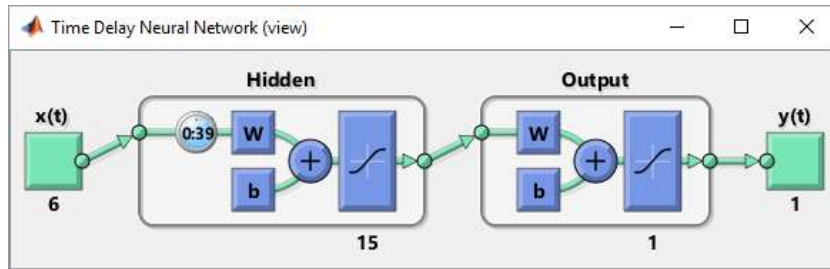


Figure 5.9 ANN structure – feed forward, time delay, tansig transfer function in output layer.

5.2 ANN training results

Because of the physical background and experimental results, I decided to set the time-delay n to 40 (dly) and number of neurons in hidden layer ($nnum$) to 15, [39], [40]. Because of satisfactory results, I decided to use less complex TDNN for the time-series problem.

When the structure and functions in ANN were chosen, other artificial neural networks with the same structure but different input sets were trained.

The trained neural networks (discussed further) differ in complexity of training sets and their data division during the training. The first group of trained neural networks uses the purelin transfer function in output layer (Figure 5.8). The second one uses tansig transfer function (Figure 5.9).

We used three different training sets. The first set contains only one transmission between staying still and walking. The second training set contains three state changes. The last one training set contains many changes between the two states, for which we supposed the best performance after the training. For each training set, we trained the ANN with both, the random and block data division. The random data division divides whole input set into training, validation and test set randomly while the block division divides the input set sequentially. The training set and the test set are disjoint (test data is not included in the set that is used for pure training).

The $plotresponse(t,y)$ MATLABTM function takes the target set t and the ANN output y , and plots them on the graph showing the errors between them. In the following graphs of the time-series response (Figure 5.11 for example), the blue curve represents a pure training set, green curve represents a validation set and red one represents a test set. At the bottom you can find a graphical representation of the error that occurs in the output of the trained ANN. The $plotregression(targets,outputs)$ MATLABTM function plots the linear regression of the targets relative to the ANN outputs, as it is shown in Figure 5.13 for example. It clearly shows the deviations of the ANN outputs and the character of the deviations related to the training outputs. The regression value R closer to the value 1 indicates the better adaptation of the trained ANN (to the training set).

The *ploterrhist(e)* MATLAB™ function plots a histogram of error values e , see relation (5.2.1) where t represents required targets from the training set and y represents the ANN outputs.

$$e = t - y \tag{5.2.1}$$

As the performance function (see performance functions) the MSE was used. It follows that lower performance means higher accuracy of evaluated results. The result “-1” means that the IMU “stays still” and “+1” means that it “walks”.

5.2.1 Results of TDNNs with purelin transfer function in output layer

As it was noted above, these networks use purelin transfer function in the output layer. The first training set (see Figure 5.10) shows approx. 22 minutes of walk and 22 minutes of staying still with the IMU in the hand. The changes in acceleration during “still” phase were caused by orientation changes of the IMU.

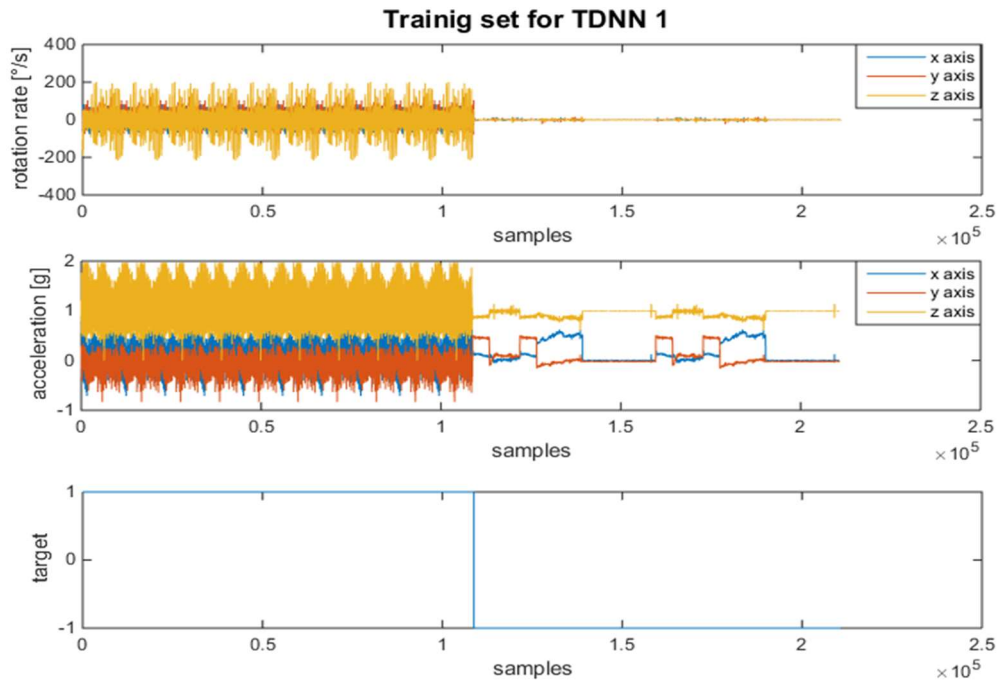


Figure 5.10 Training set for TDNN 1.

The **TDNN_divideblock_1** was trained until 15 validation checks errors had occurred, after 140 iterations. The achieved performance was 0.00644 and it took almost 5 hours. The **TDNN_dividerand_1** training stopped after 215 iterations when the performance of training was 0.00787 and it was almost stable for last 200 iterations and the duration of training ANN was almost 8 hours.

It is clear that the training set TDNN1 is not a good example of the training set for required ANN. Nevertheless, it shows what happens when the change of the state occurs very rarely. In basic, rare changes in state are not a problem for the ANN simulation. However we need the network to recognize frequent changes accurately. The ANN easily recognizes the static state (only walking, only staying still). The difficulty is to recognize the exact moment when the state change occurs. Despite of having good performance value regardless the data division, the ANN trained by this input set is not applicable in this case.

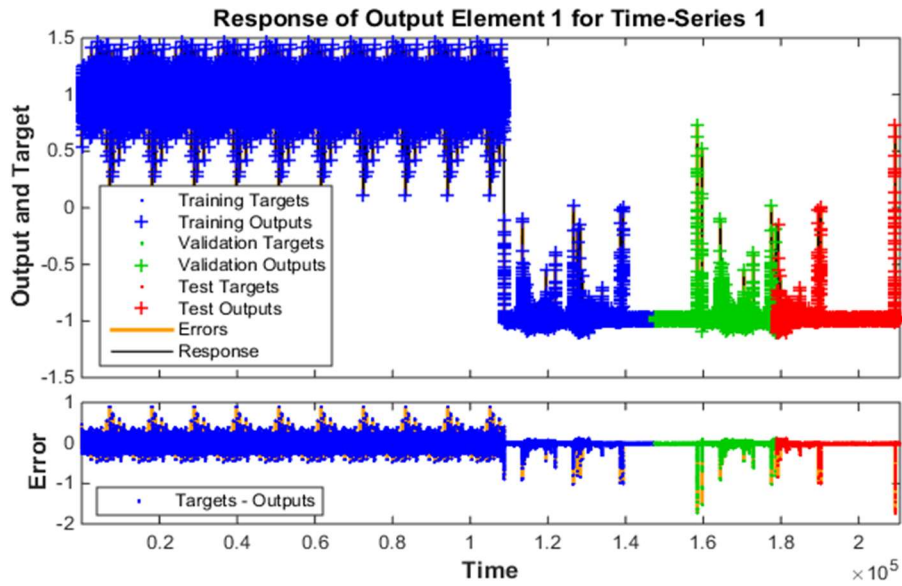


Figure 5.11 Time-series response, epoch 125, TDNN_divideblock_1.

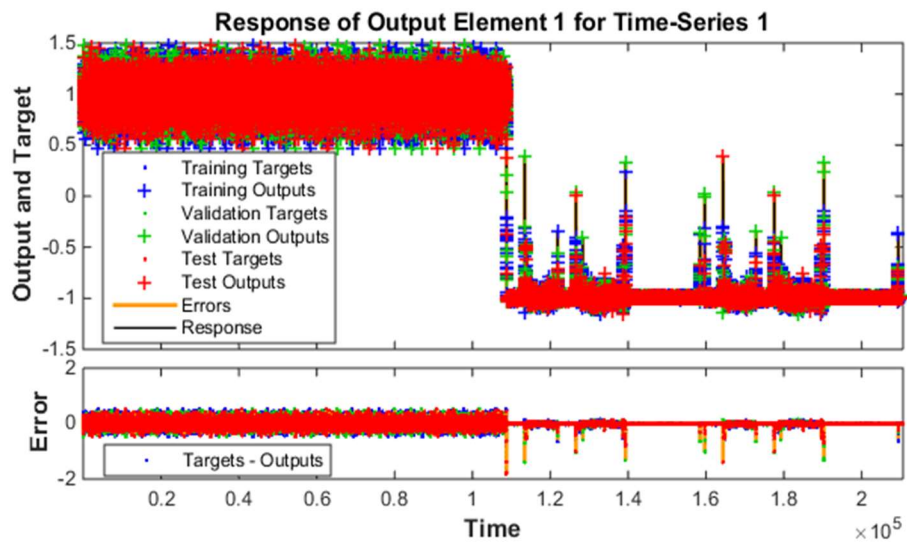


Figure 5.12 Time-series response, epoch 214, TDNN_dividerand_1.

The performance of trained network TDNN_divideblock_1 is 0.00644, though as the result seems to be very good, in validation set and test set, only the one state (“-1”) of IMU is present and for this reason the performance is really good. The performance of trained network TDNN_dividerand_1 is about 0.00787 and the reason is that the ANN recognize the static state easily. The only one state change is present and also in validation and test set such a dynamic change occurs only once in whole data set. Figure 5.13 and Figure 5.14 show the linear regression of the test set of both trained networks. Figure 5.15 and Figure 5.16 show the error histogram of the test set of both trained networks.

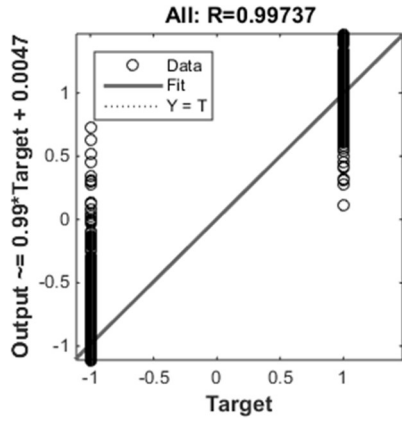


Figure 5.13 Training regression, epoch 125, TDNN_divideblock_1.

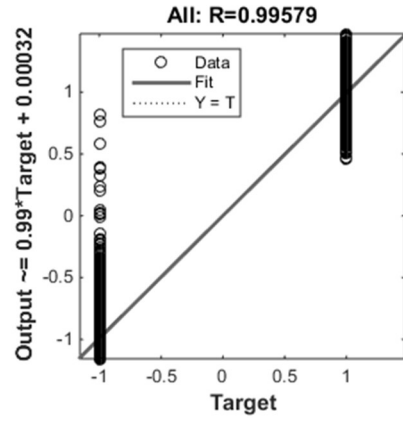


Figure 5.14 Training regression, epoch 214, TDNN_dividerand_1.

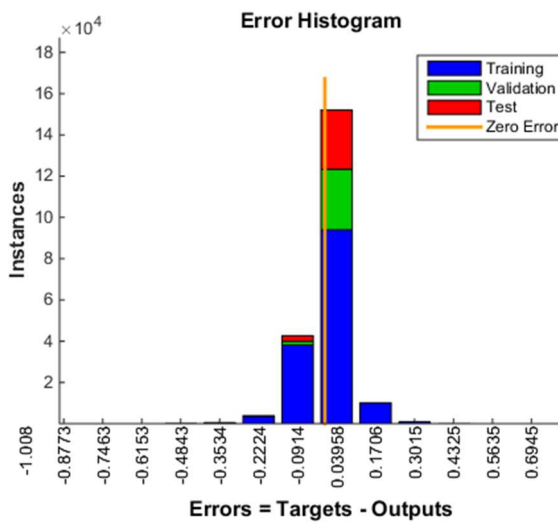


Figure 5.15 Error histogram, epoch 125, TDNN_divideblock_1.

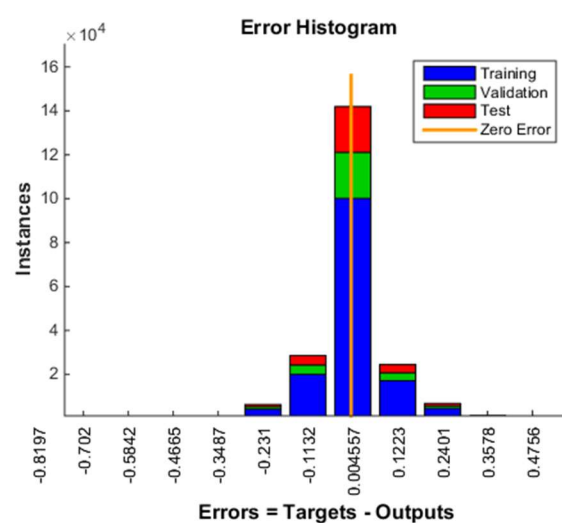


Figure 5.16 Error histogram, epoch 214, TDNN_dividerand_1.

This Chapter continues with the results of the training of the same type of neural network. However, particular ANNs differ in training set – the number of state changes. This is important if we want to find out which data are suitable for the ANN training. Resulting figures and graphs are depicted consecutively in order time-series responses, training regressions and error histograms.

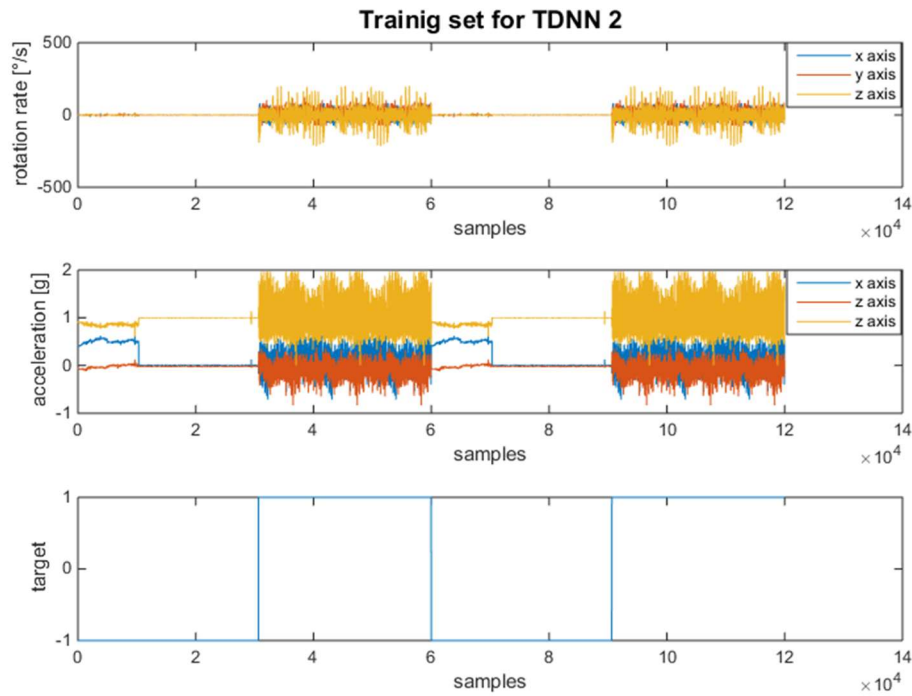


Figure 5.17 Training set for TDNN 2.

The training set data for TDNN 2 is shown in Figure 5.17. It contains three state changes and it is composed of the staying still (the IMU was hold in the hand for approx. 2 minutes, then it was laying on the table for approx. 4 minutes) and walking (the was hold in the hand while walking for approx. 6 minutes). This repeats. The **TDNN_divideblock_2** was trained for 1000 iterations, with the best performance in 999th epoch. The duration of training was almost 6 hours and the performance achieves 0.0069. **TDNN_dividernad_2** reached required performance (less than 0.001) after 86 iterations and it took almost 30 minutes.

The time-series responses are shown in Figure 5.18 and Figure 5.19. Again, in case of the block division of training, validation and test data, the validation and test set contains only one stat. That is, again, the reason why the performance achieves such a good value. The ANN is satisfactory trained for static data and does not meet the requirement for a good reactions when the state changes.

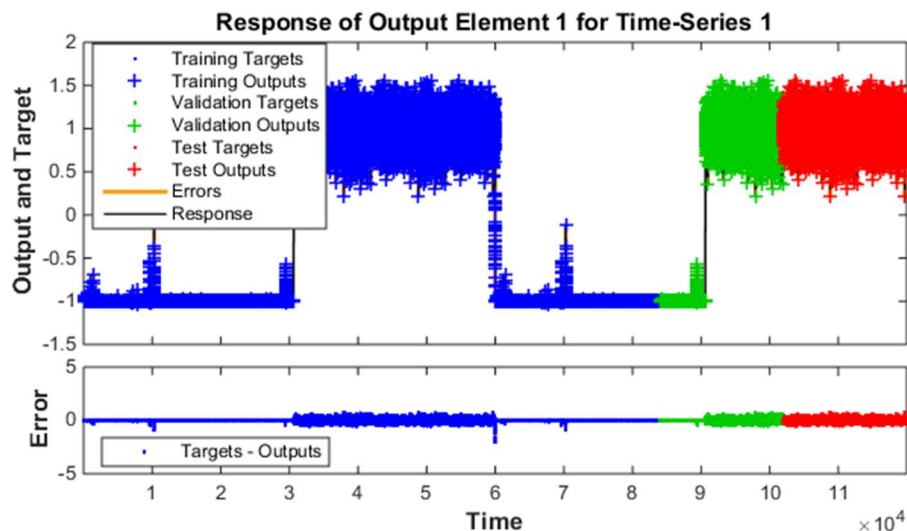


Figure 5.18 Time-series response, epoch 999, TDNN_divideblock_2.

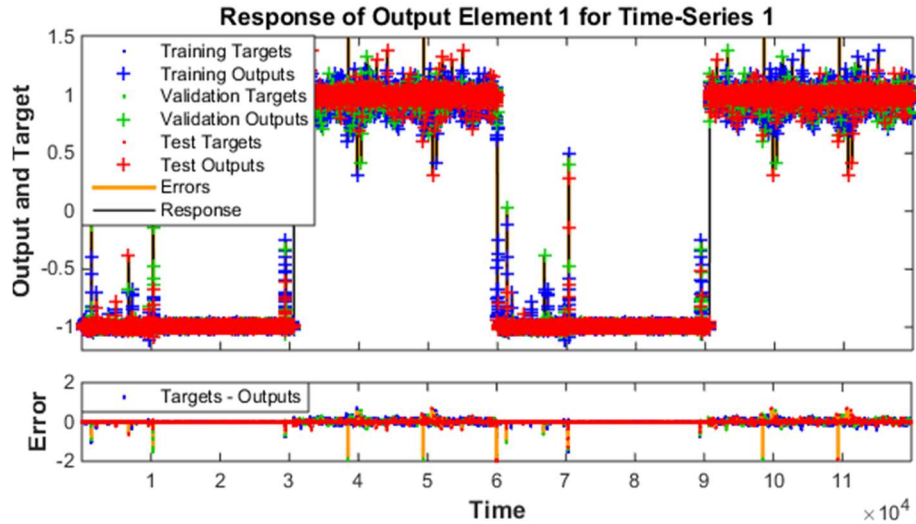


Figure 5.19 Time-series response, epoch 86, TDNN_dividerand_2

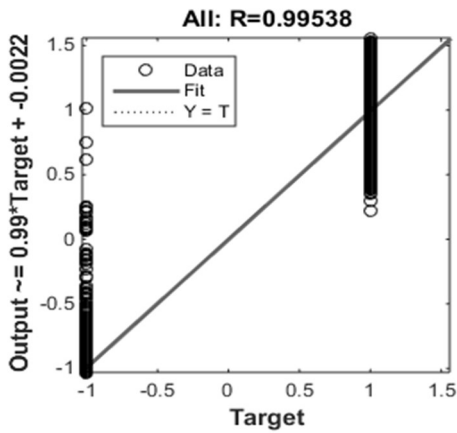


Figure 5.20 Training Regression, epoch 999, TDNN_divideblock_2.

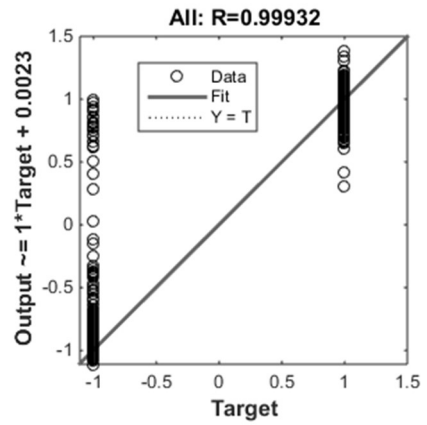


Figure 5.21 Training Regression, epoch 86, TDNN_dividerand_2.

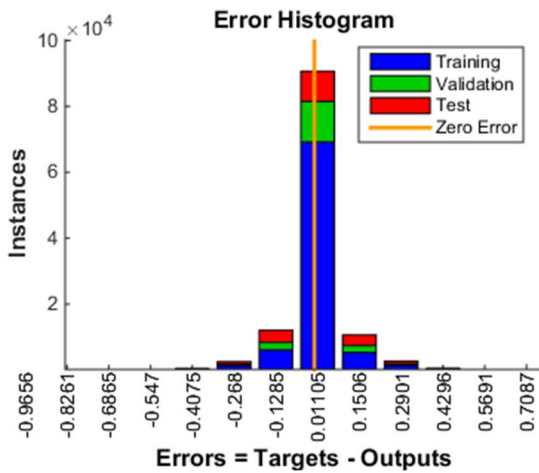


Figure 5.22 Error histogram, epoch 999, TDNN_divideblock_2.

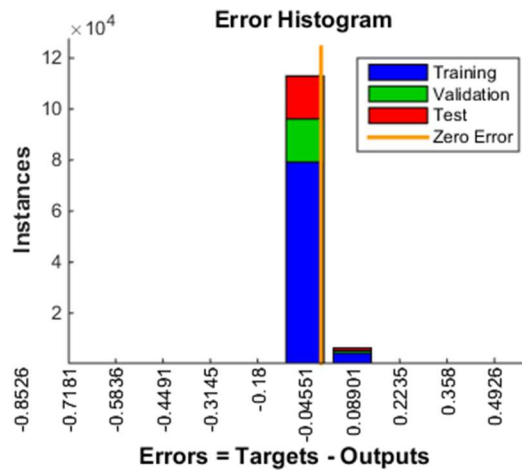


Figure 5.23 Error histogram, epoch 86, TDNN_dividerand_2.

The best resulting ANN is trained with training set that contains many state changes (Figure 5.24). Such situation occurs when the unit “walks” and stops often. Previous networks are suitable to be used when the unit is switched on but is staying still, or when it constantly moves. Further network trainings should be applied in case that the state changes occur more often. This network is also listed in our INS and it may be used for state determination.

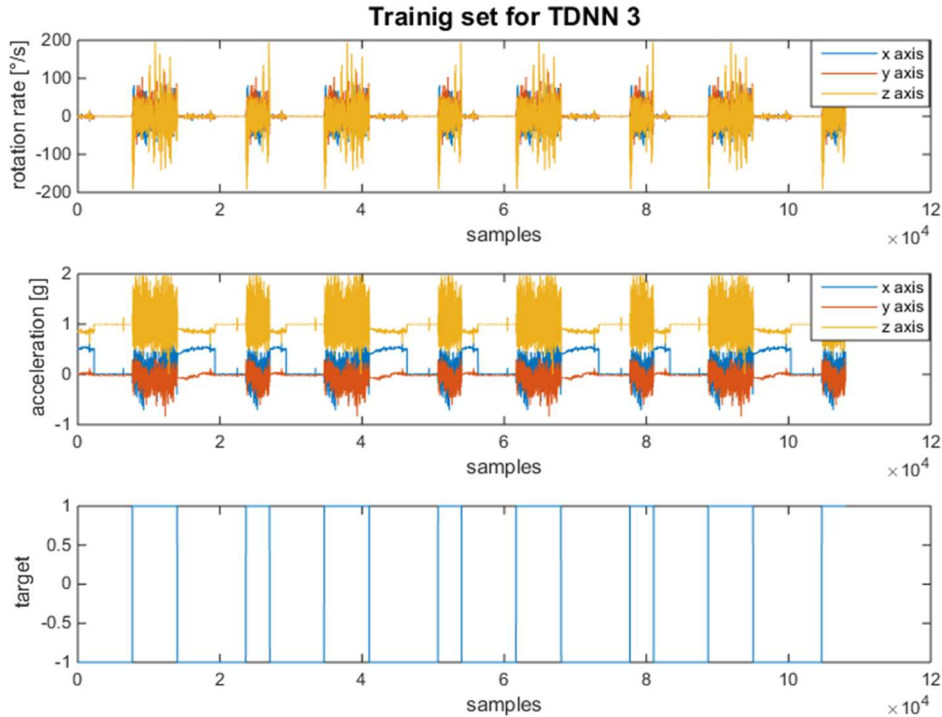


Figure 5.24 Training set for TDNN 3.

The **TDNN_divideblock_3** was trained until the performance request was met. The performance achieved the value of 0.000999 after 367 iterations, the duration of the training was almost 2 hours. The training of the **TDNN_dividerand_3** finished because of the same reason. The performance achieved the value of 0.000997 after 81 iterations, the duration of the training was 1.5 hours. Here the validation and test set contains both the static data and the state changes (3 state changes for validation set, 2 state changes for test set). To meet the performance request, the ANN must react satisfactorily also for the dynamic state changes.

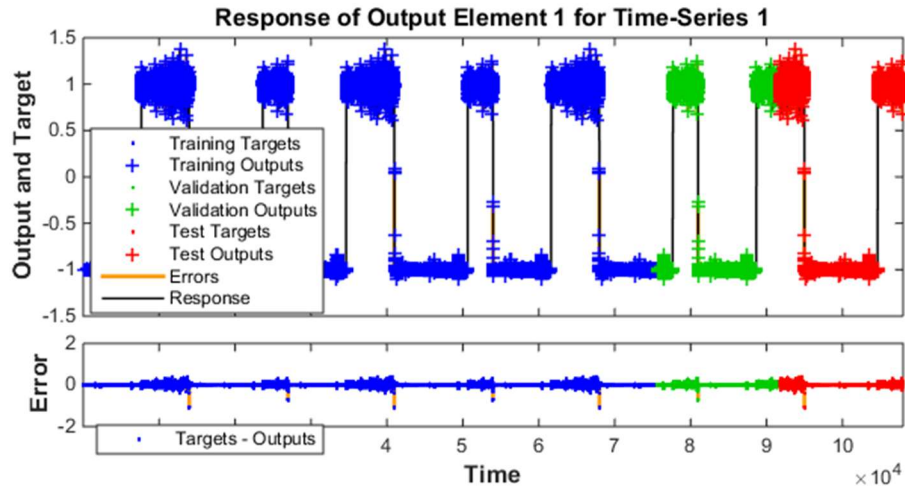


Figure 5.25 Time-series response, epoch 366, TDNN_divideblock_3.

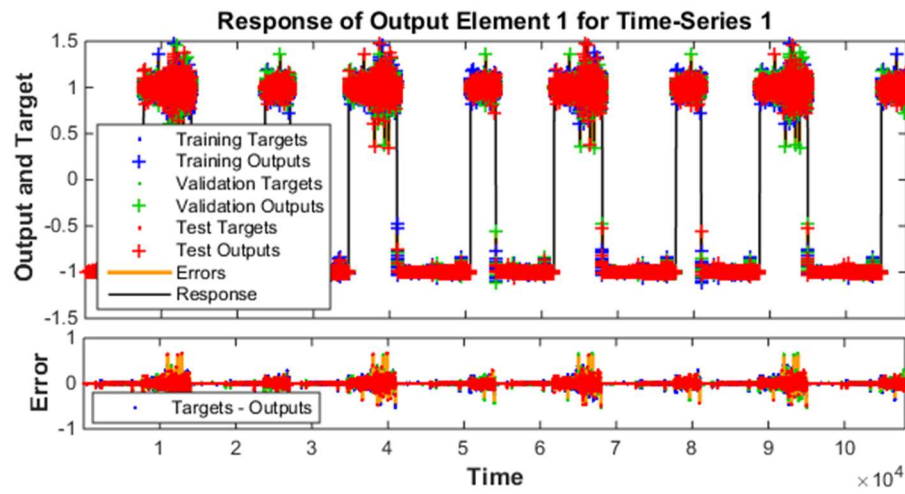


Figure 5.26 Time-series response, epoch 81, TDNN_dividerand_3.

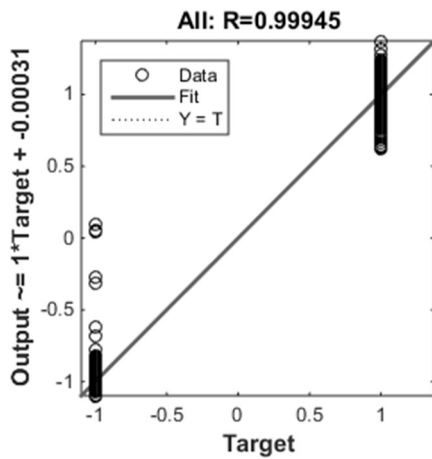


Figure 5.27 Training Regression, epoch 366, TDNN_divideblock_3.

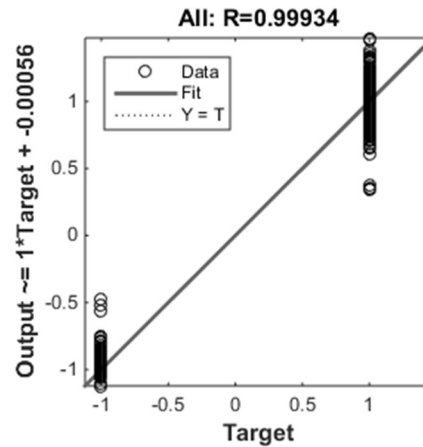


Figure 5.28 Training Regression, epoch 81, TDNN_dividerand_3.

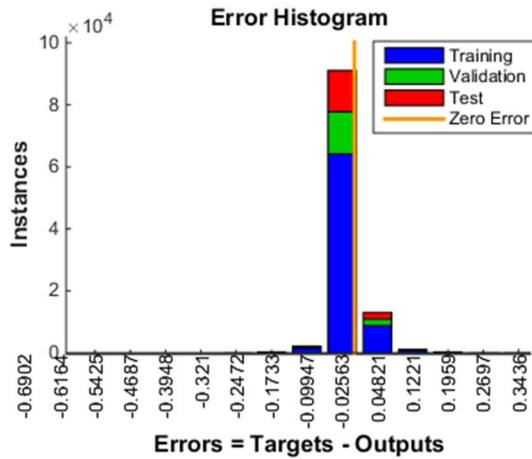


Figure 5.29 Error histogram, epoch 81, TDNN_dividerand_3.

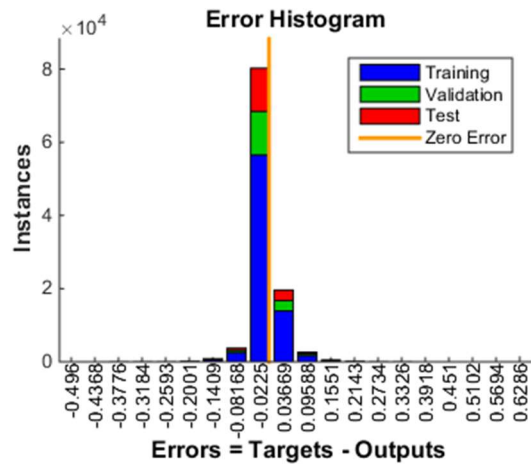


Figure 5.30 Error histogram, epoch 366, TDNN_divideblock_3.

The neural network gives the output value as a reaction on time series data. This output is more or less approaching the value +1 or the -1. This is, as noted above, because in this neural network two states are recognized, walking and staying still. This output must be further filtered (LKF and then hard limit filtration is used, see Chapter 2.8).

By adding other types of movement (measured data and appropriate targets), artificial neural network may recognize more types of motion. The most important is to recognize the staying still state as precisely as possible. The staying recognition enriches the accuracy of the dead reckoning (inertial navigation) most significantly. Obviously the accuracy is not improved when the IMU state does not change to “still”.

The duration, final number of epochs and achieved performance of the particular networks trainings are shown in TABLE 5.2. In addition, the reliability is shown as a complement of the performance to 1.

TABLE 5.2 Comparison of networks – training parameters.

Name of ANN	Duration* [h]	Epochs	Performance	Reliability
TDNN_divideblock_1	4:49:57	140	0.006440	0.993560
TDNN_dividerand_1	7:50:28	215	0.007870	0.992130
TDNN_divideblock_2	5:48:42	1000	0.006900	0.993100
TDNN_dividerand_2	0:29:49	86	0.000940	0.999060
TDNN_divideblock_3	1:52:03	367	0.000999	0.999001
TDNN_dividerand_3	0:27:15	81	0.000997	0.999003

* PC: 16 GB RAM, Intel® CORE™ 4 x i5-4690K CPU @ 3.50GHz, SSD, using MATLAB™ (used for all following experiments)

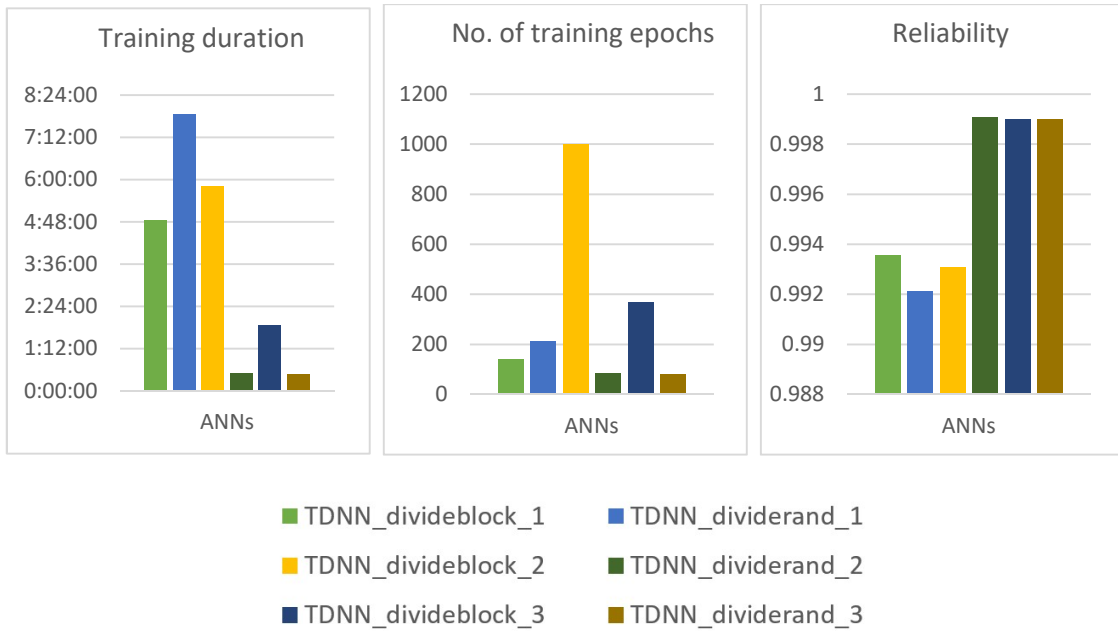


Figure 5.31 Graphical comparison of trained feed forward time delayed networks.

5.2.2 Results of TDNNs with tansig transfer function in output layer

The Chapter shows results when the tansig transfer function is used in both, hidden and output layers. The requirement for the performance is higher, because the output of the trained ANN is limited to the interval $<-1; 1>$. Thus the MSE parameter is logically lower and the performance of about 0.001 is easily achievable. Thus the performance goal value was set to 10^{-21} .

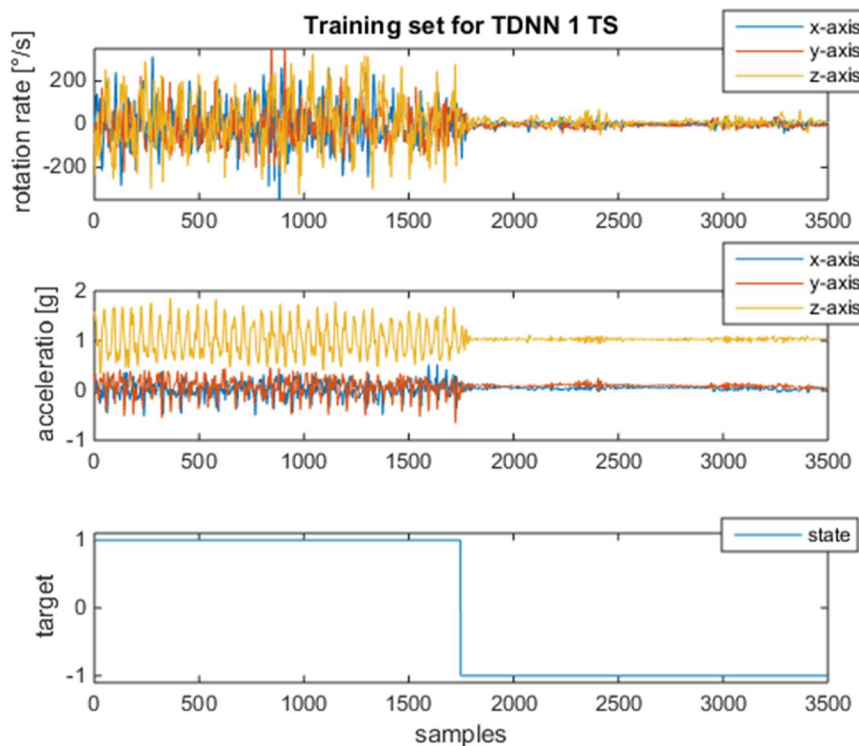


Figure 5.32 Training set for TDNN 1 TS.

The training set is almost the same as in the Chapter 5.2.1; it differs in the number of samples. Names of these ANNs were changed to be distinguished from the ANNs with linear transfer function. Further network names are then supplemented by TS (TanSig). The training set TDNN 1 TS is shown in Figure 5.32.

Following figures show the results of the neural network training while the training set TDNN 1 TS was used:

The **TDNN_divideblock_1_TS** was trained for approx. 1.5 seconds and the performance reached the required value after 29 iterations. The **TDNN_dividerand_1_TS** was trained for approx. 0.45 s and the performance reached the value after 27 iterations. The performance is very good nevertheless the same situation as in the Chapter 5.2.1 occurs. Only one change in the state is present and the ANN reacts on the “static” state.

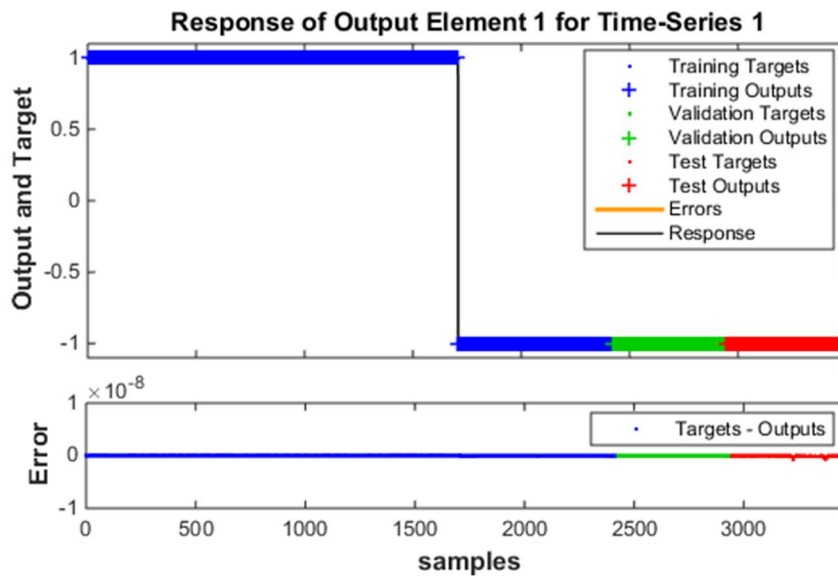


Figure 5.33 Time-series response, epoch 29, TDNN_divideblock_1_TS.

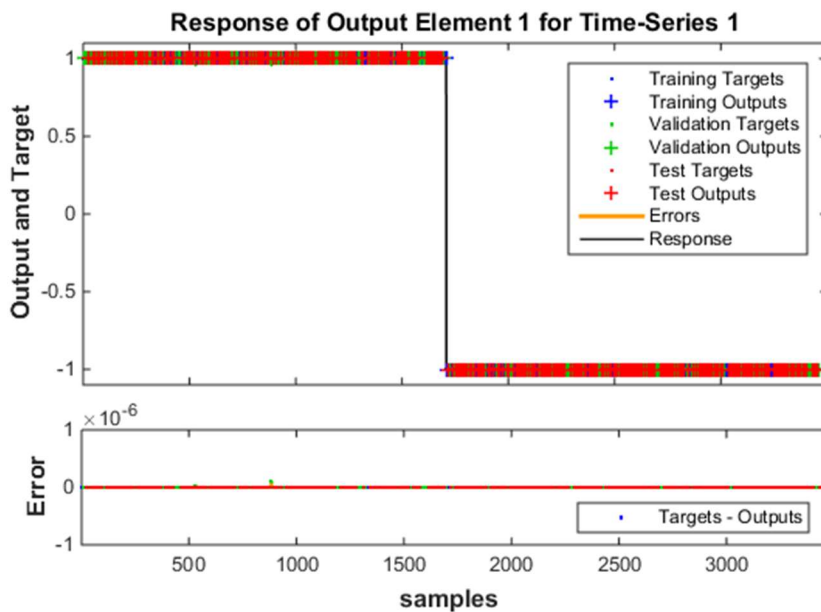


Figure 5.34 Time-series response, epoch 27, TDNN_dividerand_1_TS.

The performance of trained network TDNN_divideblock_1_TS is $6.23 \cdot 10^{-22}$, though as the result seems to be very good, in validation set and test set, only the one state (“-1”) of IMU is present. The performance of trained network TDNN_dividerand_1_TS is about $9.77 \cdot 10^{-22}$ and the reason is that the ANN recognizes the static state easily. The only one state change is present in whole input set.

Figure 5.35 and Figure 5.36 show the linear regression of the test set of both trained networks. Figure 5.37 and Figure 5.38 show the error histogram of the test set of both trained networks.

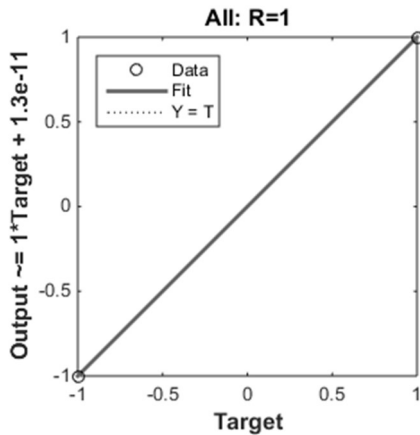


Figure 5.35 Training regression, epoch 29, TDNN_divideblock_1_TS.

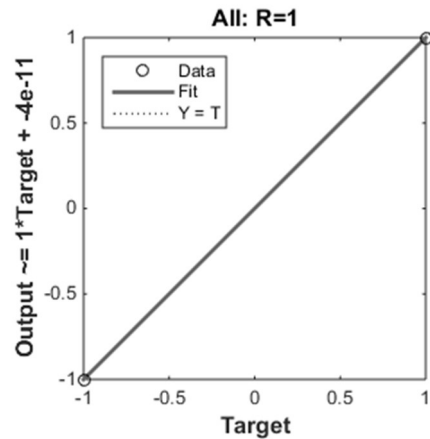


Figure 5.36 Training regression, epoch 27, TDNN_dividerand_1_TS.

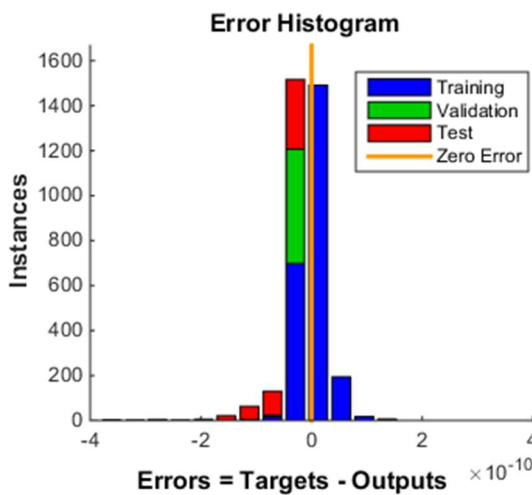


Figure 5.37 Error histogram, epoch 29, TDNN_divideblock_1_TS.

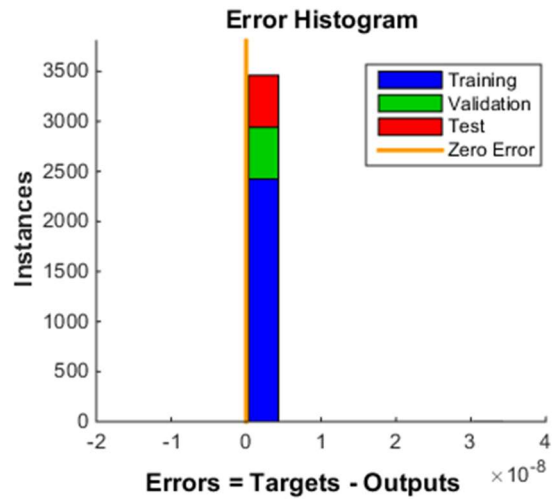


Figure 5.38 Error histogram, epoch 27, TDNN_dividerand_1_TS.

This Chapter continues with the results of other trained neural networks with the tansig transfer function in the output layer. They have the same structure and properties, but their input sets differ in the number of changes of the state.

Figures and graphs are depicted in the same order as for the input set TDNN 1 TS. The input set TDNN 2 TS is shown in Figure 5.39.

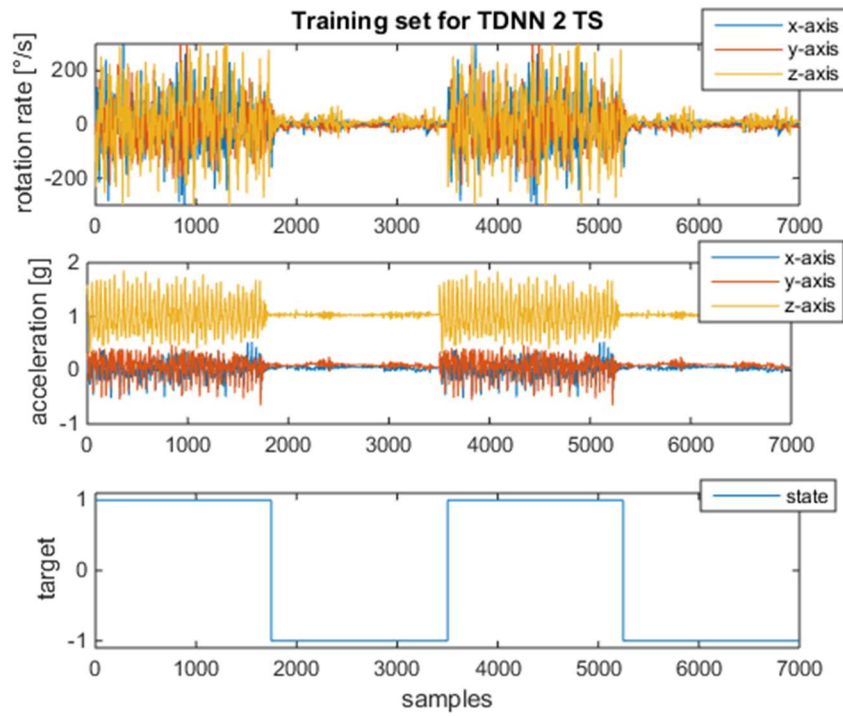


Figure 5.39 Training set for TDNN 2 TS.

Further figures show the results of the neural network trainings with the input set TDNN 2 TS:

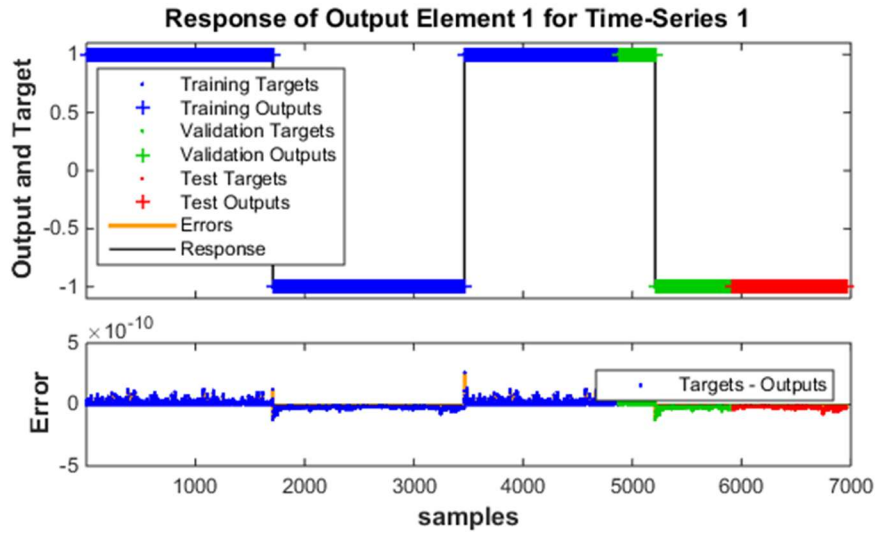


Figure 5.40 Time-series response, epoch 28, TDNN_divideblock_2_TS.

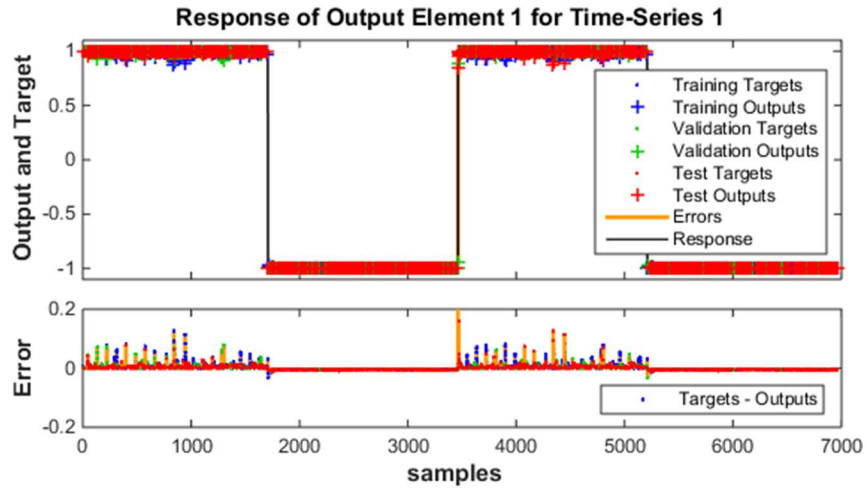


Figure 5.41 Time-series response, epoch 28, TDNN_dividerand_2_TS.

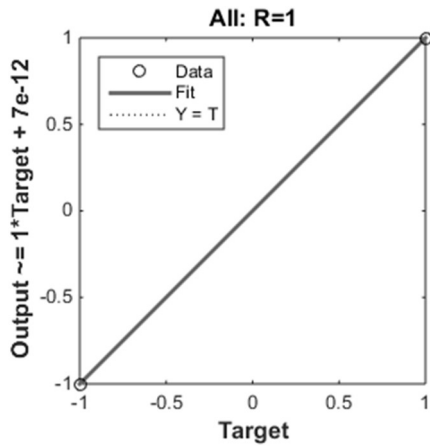


Figure 5.42 Training Regression, epoch 28, TDNN_divideblock_2_TS.

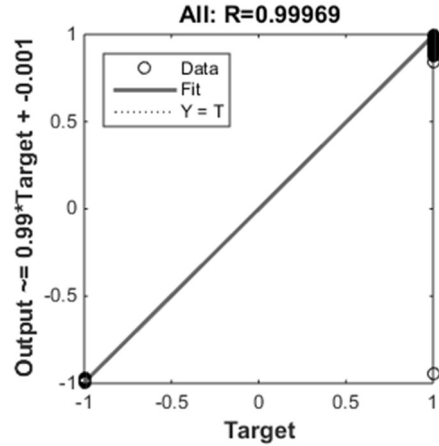


Figure 5.43 Training Regression, epoch 28, TDNN_dividerand_2_TS.

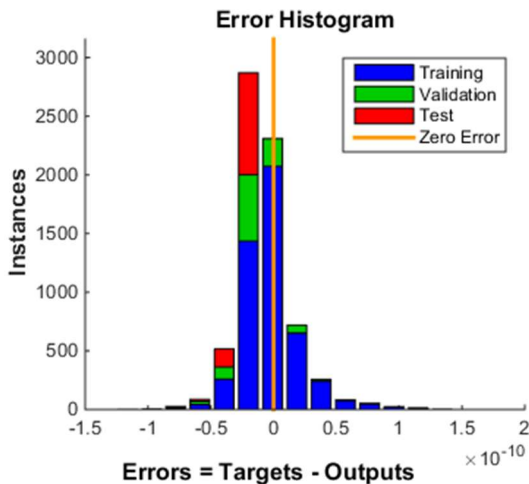


Figure 5.44 Error histogram, epoch 28, TDNN_divideblock_2_TS.

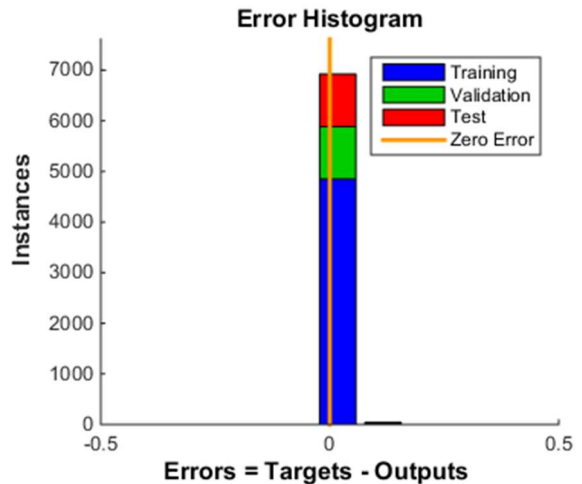


Figure 5.45 Error histogram, epoch 28, TDNN_dividerand_2_TS.

The TDNN_divideblock_2_TS was trained for 71 seconds, with achieved performance of $5.62 \cdot 10^{-22}$. The validation set contains one state change and the test set does not

contain any change in state. The **TDNN_dividernad_2_TS** also reached the required performance after 2 minutes with 28 iterations. Thus the very high performance request was satisfied after only a few iterations.

The training set TDNN 3 TS is shown in Figure 5.46. Further figures show the results of the ANN training with the input set TDNN 3 TS.

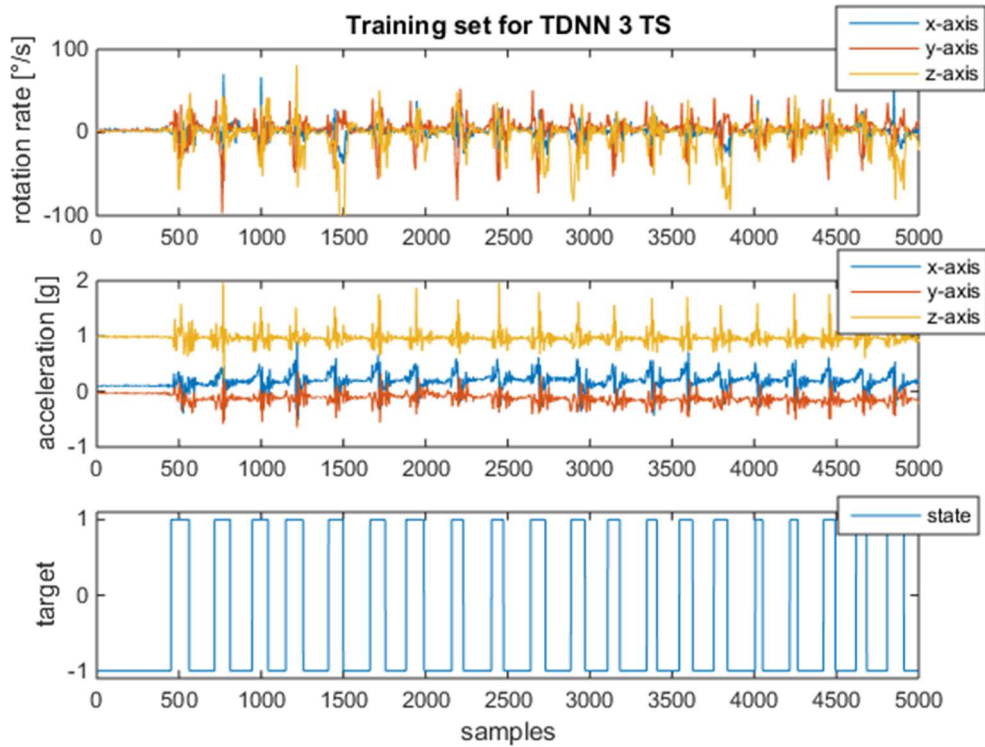


Figure 5.46 Training set for TDNN 3 TS.

The **TDNN_divideblock_3_TS** was trained for almost 15 minutes, with the achieved performance of 0.00187 after 90 iterations. The training process stopped because the Mu maximum value (see TABLE 5.1) was reached.

The **TDNN_dividernad_3_TS** was trained for approx. 10 minutes and the training process stopped after 73 iterations when the Mu parameter reached the limit. The performance reached the value of 0.00229.

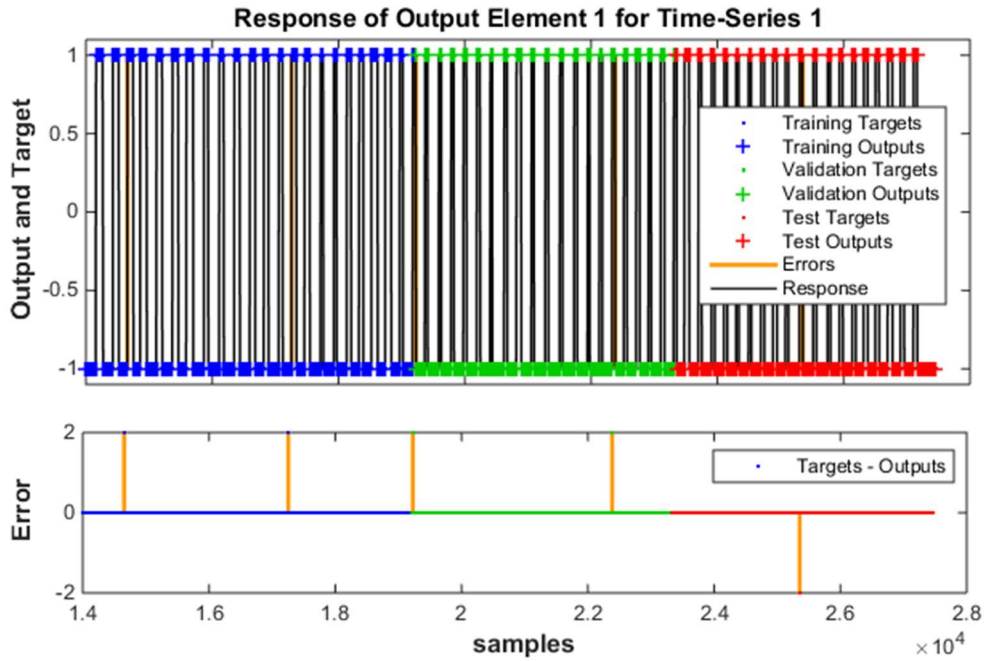


Figure 5.47 Time-series response, epoch 90, TDNN_divideblock_3_TS.

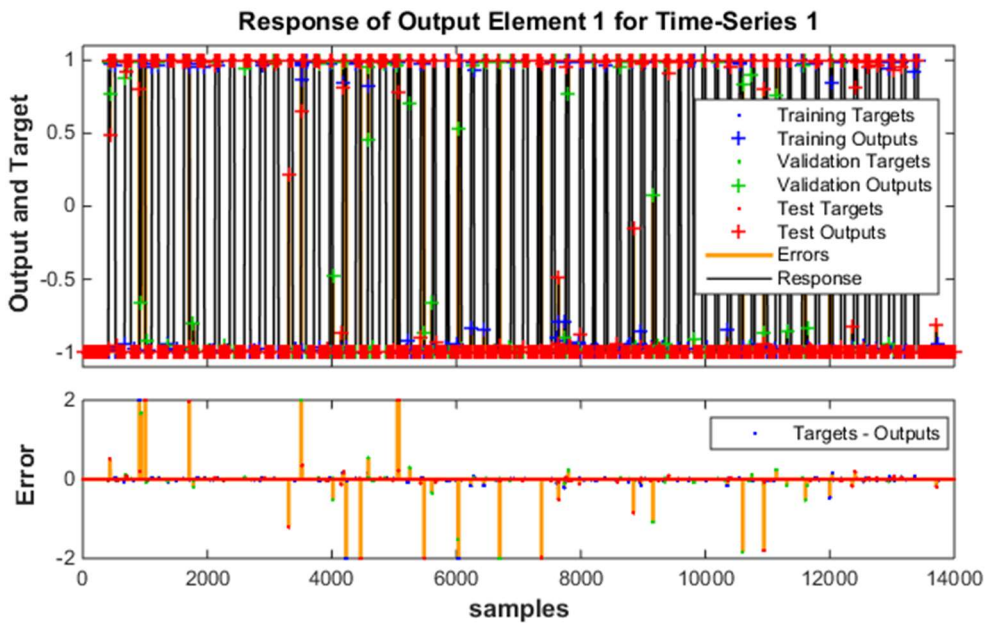


Figure 5.48 Time-series response, epoch 73, TDNN_dividerand_3_TS.

The validation set contains a lot of state changes and so does the test set. Thus the performance does not reach so good values. Nevertheless, changes in state are recognized with the best performance. Those trained networks are the most suitable neural networks for this kind of tasks. Because of the walking is dynamic process and parameters taken from the sensors may fluctuate or differ in time, the TDNN_dividerand_3_TS is used in the IMU by default. There is also possibility to change the used ANN to TDNN_divideblock_3_TS.

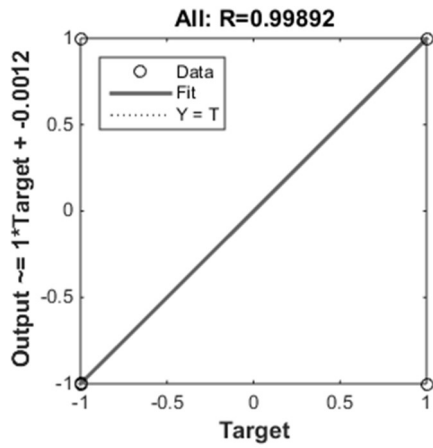


Figure 5.49 Training Regression, epoch 90, TDNN_divideblock_3_TS.

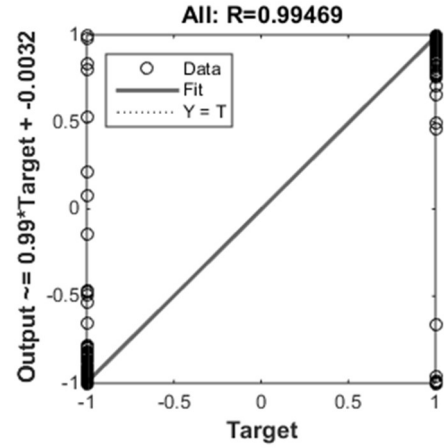


Figure 5.50 Training Regression, epoch 73, TDNN_dividerand_3_TS.

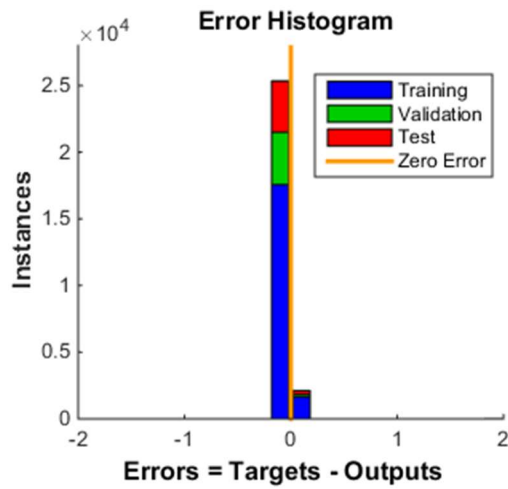


Figure 5.51 Error histogram, epoch 90, TDNN_divideblock_3_TS.

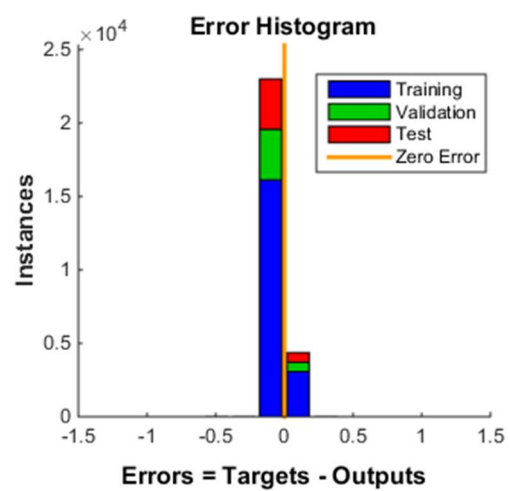


Figure 5.52 Error histogram, epoch 73, TDNN_dividerand_3_TS.

TABLE 5.3 Comparison of networks II – training parameters

Name of ANN	Duration [h]	Epochs	Performance	Reliability
TDNN_divideblock_1_TS	0:01:26	29	$6.23 \cdot 10^{-22}$	≈ 1
TDNN_dividerand_1_TS	0:00:46	27	$9.77 \cdot 10^{-22}$	≈ 1
TDNN_divideblock_2_TS	0:01:11	28	$5.62 \cdot 10^{-22}$	≈ 1
TDNN_dividerand_2_TS	0:02:00	28	$5.99 \cdot 10^{-22}$	≈ 1
TDNN_divideblock_3_TS	0:14:36	90	0.00187	0.99813
TDNN_dividerand_3_TS	0:09:46	73	0.00229	0.99771

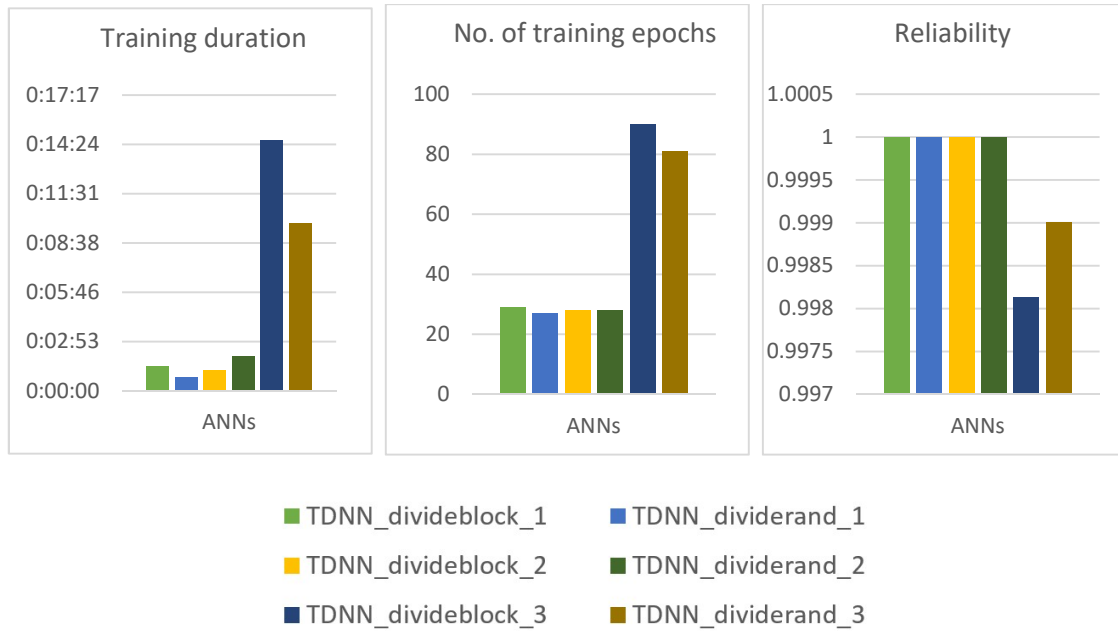


Figure 5.53 Graphical comparison of trained feed forward time delayed networks

5.3 Additional ANNs

The first additional neural network was created to demonstrate the results in short ranges without walking (ADD_TDNN_1). The structure and functions remain, however the input set was changed to data measured during IMU rotations, vibrations, oscillations, shifting etc. Output was then set to “+1” in case of any movement and “-1” in case of staying “still” in hand.

The second additional neural network was created from data representing only transitions between the two states – walk and staying still (ADD_TDNN_2). This is not the situation that usually occurs, nevertheless the network has to be trained as precisely as possible for state changes. The input set is composed of moving phase (one human step) and still phase (duration 1 s) repeated many times. The training set contains 41244 samples (record time: 491 s). A part of the collected data set is shown in Figure 5.54. Both transfer functions inside ADD_TDNN_2 were set to hyperbolic tangent sigmoid (TanSig). The result of the network training shows that the network is highly adapted for frequent state changes (see Figure 5.55). The results of this ANN are similar to the TDNN_dividerand_3_TS, however its training set was strictly determined by regular state changes.

TABLE 5.4 Training parameters of the additional networks.

name of ANN	duration [h]	epochs	performance	reliability
ADD_TDNN_1	00:47:15	87	0.01091	0.98909
ADD_TDNN_2	0:10:56	50	0.00333	0.99667

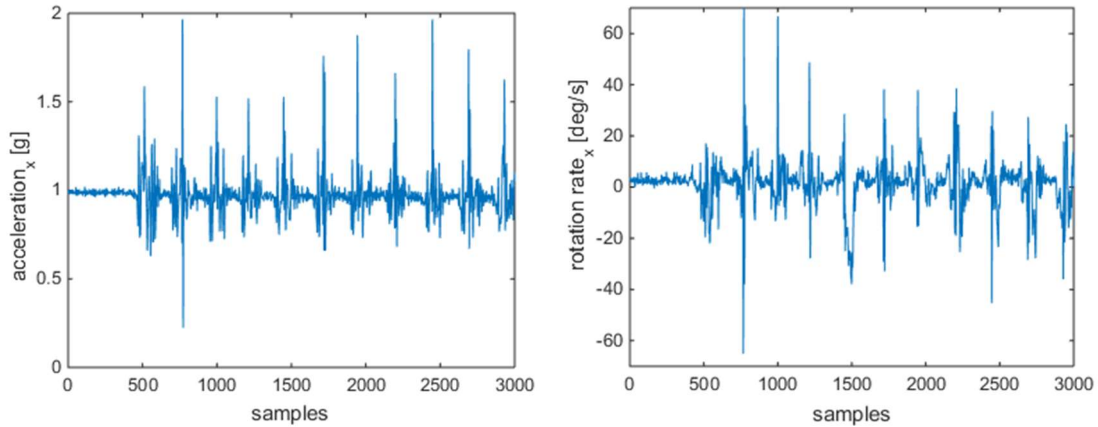


Figure 5.54 The data used as a part of the training set for ADD_TDNN_2 training.

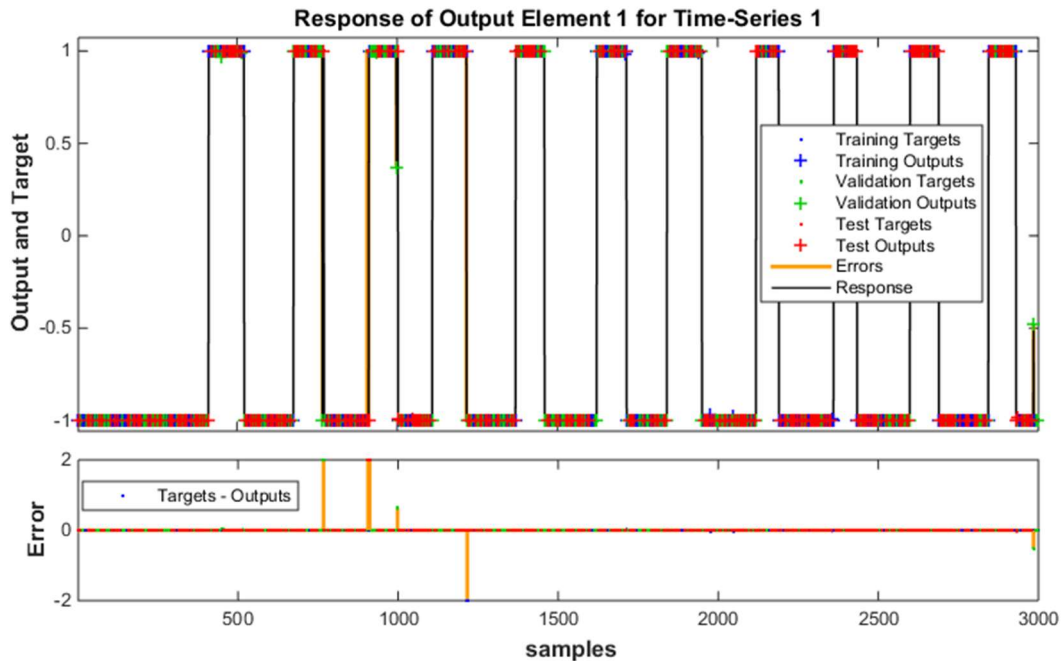


Figure 5.55 Response of the ADD_TDNN_2.

5.4 Other types of ANNs

In order to complete the comparison, the results from some other types of ANNs are shown in this Chapter. The training set for those ANNs was the same as the training set for the ADD_TDNN_2 (see data example in Figure 5.54). The division of training, test and validation data set was random in a ratio of 70:15:15.

In time delayed ANNs, the same number of history and the same size of the hidden layer was set. For the “pattern recognition and classification ANN” the reduction to 3 neurons in hidden layer was set (the first case). Due to very high inaccuracy, the hidden layer was enlarged back to 15 neurons (the second case). The results are given below. Those neural networks are not included in the final data processing software.

– **NARX TD ANN (with time delay)** – Figure 5.1 shows the principle

The structure of the NARX TD ANN shows Figure 5.56. The previous output is appended to the input of the next sample as the seventh variable. This type of ANN also solves this task very effectively, as it is clear from Figure 5.57.

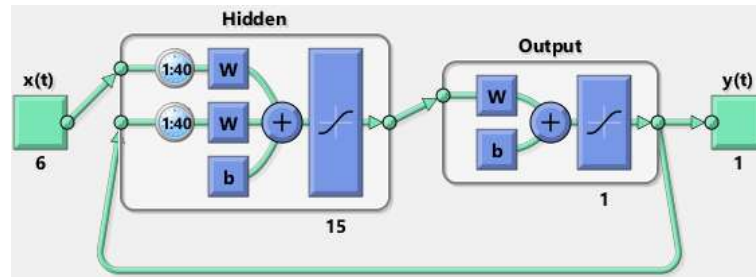


Figure 5.56 NARX TD ANN structure.

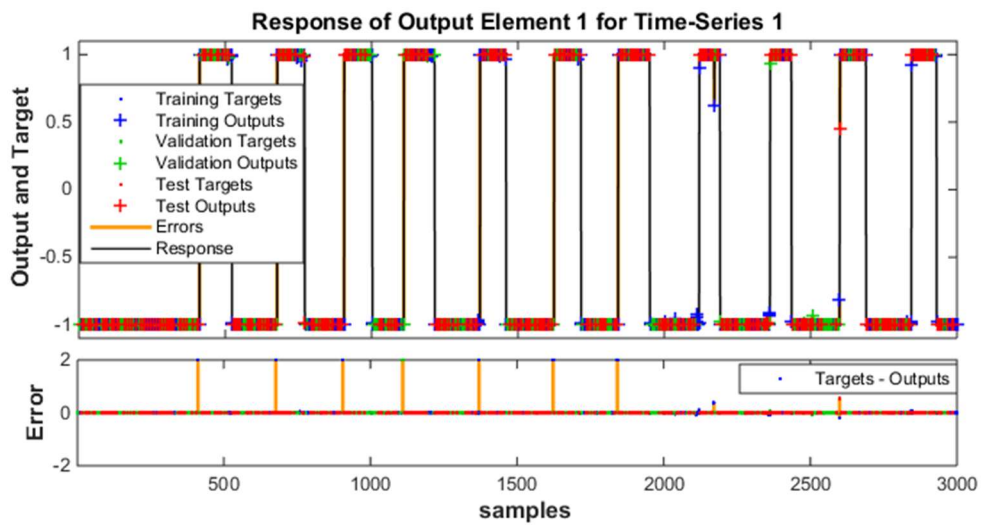


Figure 5.57 NARX TD ANN response.

Algorithms	
Data Division:	Random (dividerand)
Training:	Levenberg-Marquardt (trainlm)
Performance:	Mean Squared Error (mse)
Calculations:	MATLAB
Progress	
Epoch:	0 / 42 iterations / 1000
Time:	0:07:10
Performance:	2.01 / 0.00472 / 0.00
Gradient:	8.06 / 1.94e-05 / 1.00e-07
Mu:	0.00100 / 1.00e-10 / 1.00e+10
Validation Checks:	0 / 6 / 6

Figure 5.58 Training parameters for NARX TD ANN.

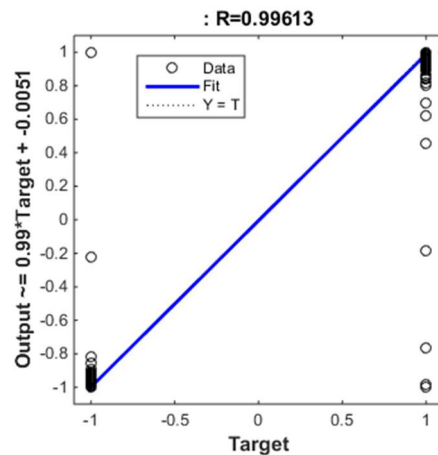


Figure 5.59 Training Regression, epoch 42.

Figure 5.58 shows the training process parameters and the regression of the trained neural network is shown in Figure 5.59. The results seem to be very good. Nevertheless, the output after the Kalman filtration is slightly worse than the TDNN_dividerand_3 and TDNN_dividerand_3_TS output after the Kalman filtration. In addition, the NARX TD ANN training and processing is more time consuming due to its larger input matrix.

– **Feed forward ANN (FF ANN without time delay)**

This type of ANN is not suitable for our dynamic problem because its input set is static. It reacts to six input variables that were taken from sensors in a single instance and it is not able to recognize the walk and “still” phase accurately.

The structure of FF ANN is shown in Figure 5.60 and the time series response is shown in Figure 5.61. The training parameters (Figure 5.62) and training regression (Figure 5.62) are available below.

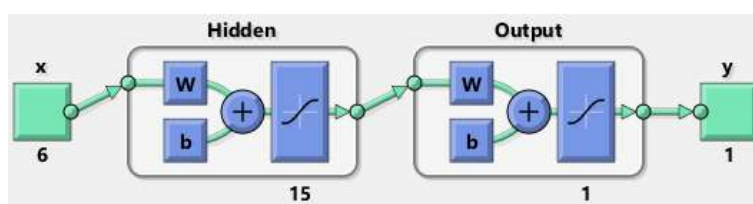


Figure 5.60 FF ANN structure (without time delay).

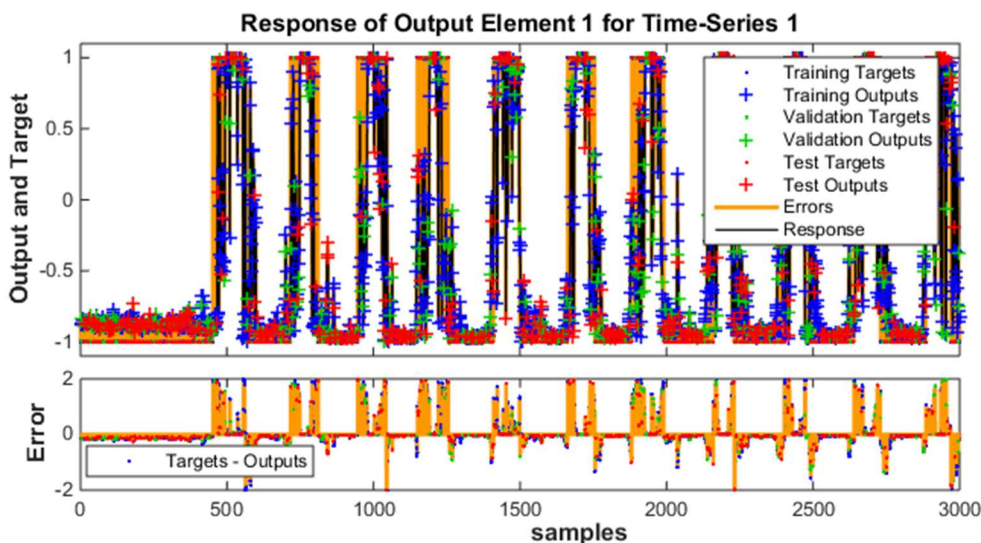


Figure 5.61 FF ANN response (without time delay).

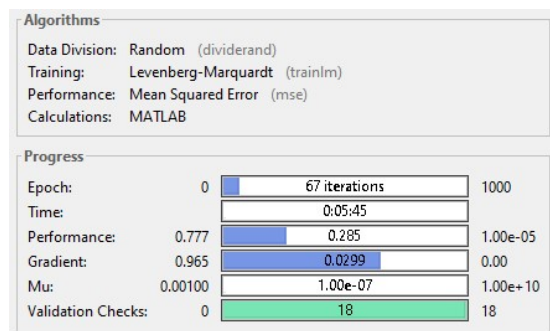


Figure 5.62 Training parameters for FF ANN

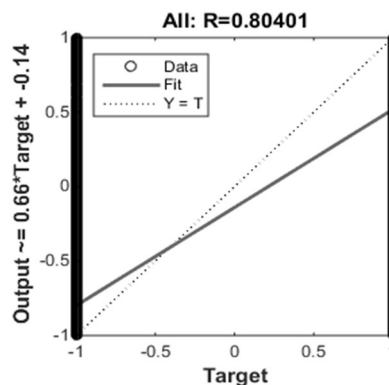


Figure 5.63 Training Regression, epoch 67.

– **NARX ANN (without time delay)**

NARX ANN without time delay structure is shown in Figure 5.64. The time series response (Figure 5.65) shows a very clear decision on the IMU state, this type of ANN may also be used for given task with further processing. The time and memory consumption is decreased because of zero history length in the input (time delay), thus the training process is not so much time consuming. The issue here is the inaccuracy on state changes, in particular cases when the status changes from “still” to walk. This is a very unintended behaviour mentioned previously. Training parameters (Figure 5.66) and training regression (Figure 5.67) are present below.

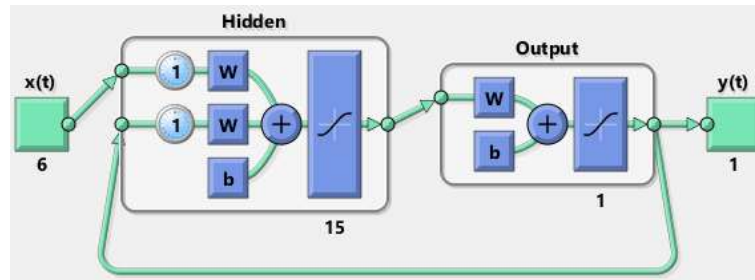


Figure 5.64 NARX ANN structure (without time delay).

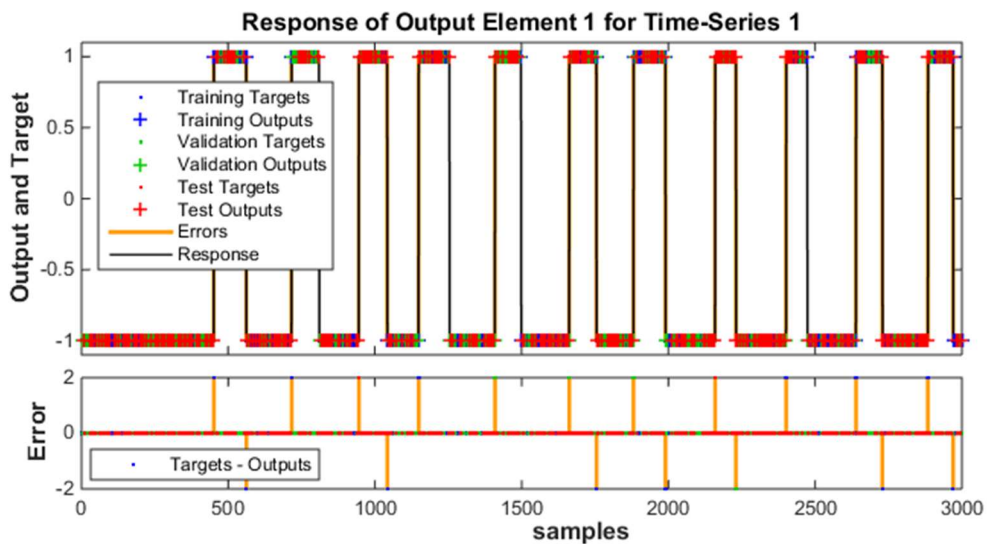


Figure 5.65 NARX ANN response (without time delay).

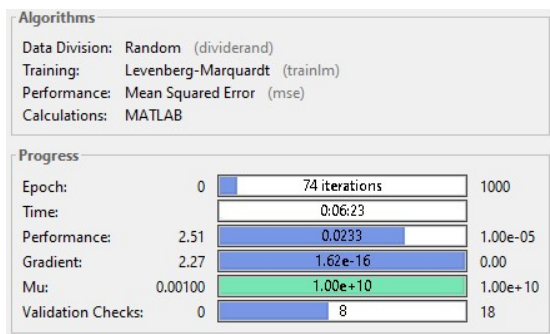


Figure 5.66 Training parameters for FF ANN.

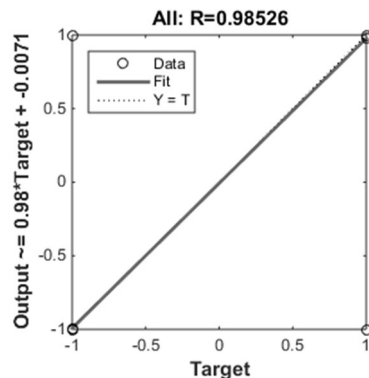


Figure 5.67 Training Regression, epoch 74.

– Pattern recognition and classification ANN – 3 hidden neurons

Pattern recognition and classification ANN (PR&C ANN) works as a classifier and divides output into two categories in our case (see Figure 5.68). The Confusion Matrix (Figure 5.70), [43], shows the correctly and incorrectly classified outputs considering to training set targets. The training parameters are shown in Figure 5.69. Further figures represent analogically results of PR&C ANN with 15 neurons in hidden layer instead of 3 neurons (see Figure 5.72 and Figure 5.71). This type of ANN is not suitable for mentioned task as it is clear from the presented results.

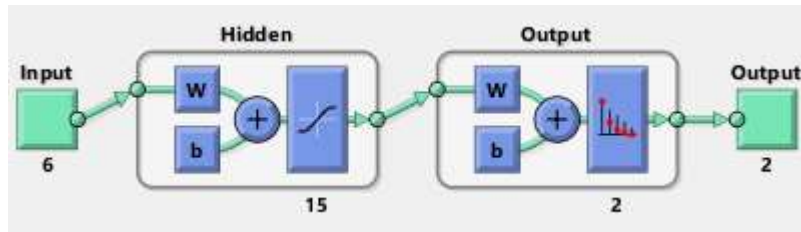


Figure 5.68 PR&C ANN structure (3 hidden neurons).

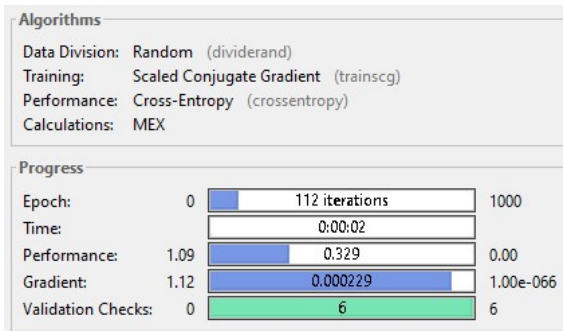


Figure 5.69 Training parameters PR&C ANN.

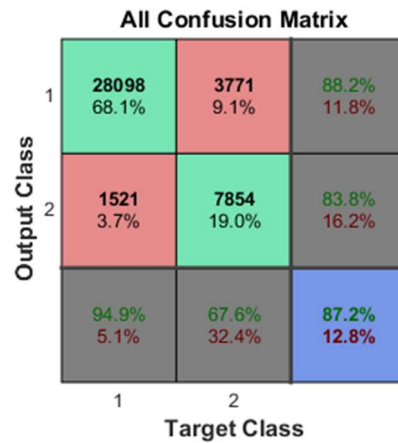


Figure 5.70 Confusion Matrix I.

Pattern recognition and classification ANN – 15 hidden neurons

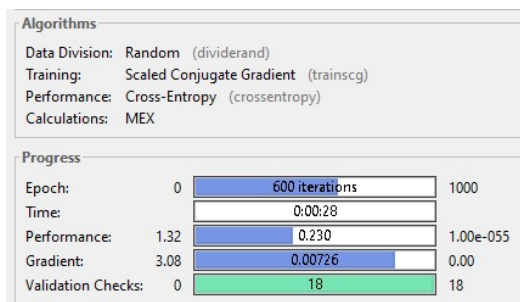


Figure 5.71 Training parameters PR&C

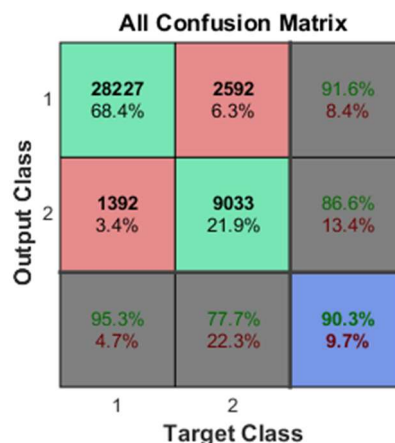


Figure 5.72 Confusion Matrix II.

TABLE 5.5 shows comparative results of all other ANNs presented in Chapter 5.4.

TABLE 5.5 Training parameters of the other types of ANNs.

name of ANN	duration [h]	epochs	performance	reliability
NARX TD ANN	0:30:43	42	0.00472	0.99528
FF ANN	0:05:45	67	0.28500	0.71500
NARX ANN	0:06:23	74	0.02330	0.97670
PR&C – 3 neurons	0:00:02	112	0.32900	0.67100
PR&C – 15 neurons	0:00:28	600	0.23000	0.77000

6 DATA PROCESSING

Thanks to the ANN we know the state of the IMU. Provided we trust it, the attitude of the IMU in time can be determined with higher accuracy. Whenever the state defined by the ANN is “still”, the accurate (absolute) tilt of the IMU can be recalculated. The only property that still remains to be absolutely determined is heading.

The determination of heading comes from magnetometer data. The value that has to be checked before heading determination is total magnetic field. If the value is too low or too high, there are other influences than Earth’s magnetic field, and the heading cannot be determined by the magnetometer. In that case we have to rely on integrated data from gyroscope.

In the second case, when the value of total magnetic field falls within the given range, the heading can be computed. However, it can be done after the acceleration data tilt compensation – derotation into the flat level. After the proper magnetometer data derotation the heading value can be determined only from x -axis and y -axis of magnetometer data. The absolute attitude of the IMU can be finally determined.

In the next part, new proposal algorithm for attitude determination is presented.

6.1 When the state is “still”

The neural network decides that the IMU state is “still”. Then, there are more ways, as described in theoretical part, how to rotate the IMU back into flat level (horizontal plane). Figure 6.1 clearly shows all three Euler stages of derotation from IMU (RPY) coordinates into inertial, ENU coordinates.

At first, the derotation of heading needs to be done by an angle $-\theta$. This rotation is solved separately after the derotation into flat position. Rotations by angles $-\Psi$ and $-\Phi$ define the tilt of the IMU in RPY coordinates and through them the IMU coordinates from RPY coordinate system into ENU coordinates can be converted.

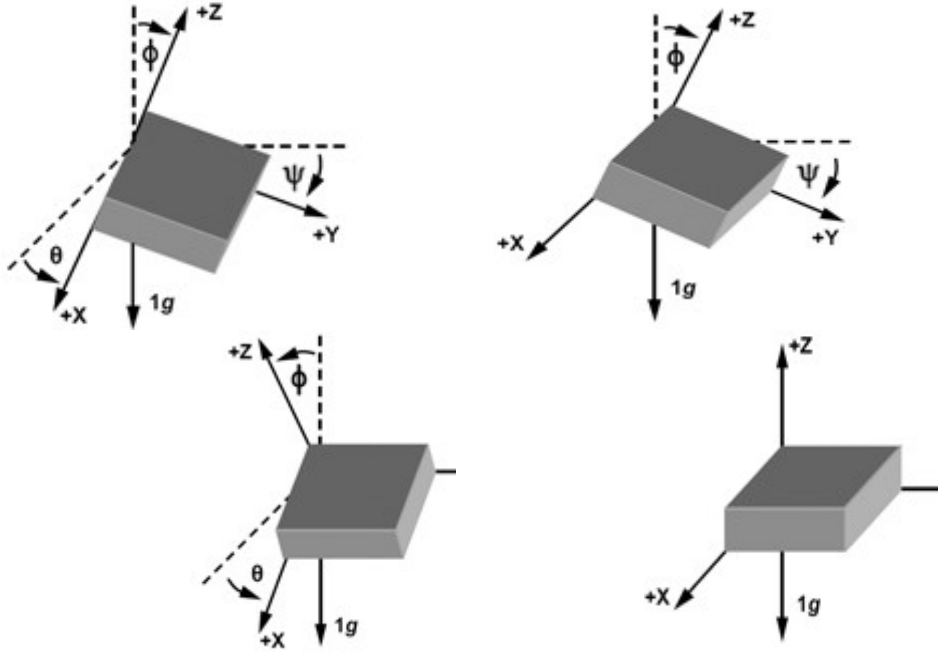


Figure 6.1 Derotation from the RPY (body) coordinates to ENU coordinates.

The Euler angles or direction cosine matrix (DCM) method seems to be easy to use. Nevertheless, the gimbal lock is the fundamental problem for the INS. For this reason the application of quaternions must be implemented. In proposed algorithms, the rotation around z -axis is denoted by symbol Ψ , the rotation around y -axis is denoted by symbol θ , and the rotation around x -axis is denoted by symbol Φ .

From accelerometers, after axis alignment, compensation and calibration, the accelerations in particular axes are obtained, the vector $\vec{a} = (acc_x, acc_y, acc_z)$ for each time step. In the source code, the calibrated accelerometer output is stored in three-dimensional vector **acc** with components **acc.x**, **acc.y** and **acc.z**. In each cycle the computation of α is performed. It expresses the angle between the acceleration vector **acc** and last acceleration vector (**accXlast**, **accYlast**, **accZlast**). The vector of the last acceleration \vec{acc}_{last} is always $(0, 0, 1)$ in order to get absolute attitude. That means it is set to the value that is present on the accelerometer output in case that the IMU is placed horizontally. The angle α is used in formulas (6.1.1) – (6.1.4) and it is based on fact that we are able to determine the quaternion, in this source code called **orientation** (Figure 6.2).

$$\alpha = \text{acos}(accXlast \cdot acc.x + accYlast \cdot acc.y + accZlast \cdot acc.z) \quad (6.1.1)$$

$$rotVecX = \frac{1}{\sin(\alpha)} \cdot (accYlast \cdot acc.z - accZlast \cdot acc.y) \quad (6.1.2)$$

$$rotVecY = \frac{1}{\sin(\alpha)} \cdot (accZlast \cdot acc.x - accXlast \cdot acc.z) \quad (6.1.3)$$

$$rotVecZ = \frac{1}{\sin(\alpha)} \cdot (accXlast \cdot acc.y - accYlast \cdot acc.x) \quad (6.1.4)$$


```

orientation = QQuaternion::fromAxisAndAngle(p, q, r, angle)
//p = rotVecX;
//q = rotVecY;
//r = rotVecZ;
//angle = -alfa*180/M_PI);
    
```

Figure 6.2 Quaternion **orientation** determination.

For further application in formulas, the quaternion **orientation** equals **ORI**.

The angle θ and φ are subsequently determined from the quaternion by formulas (6.1.5) and (6.1.6):

$$\theta = -\text{asin}(2 \cdot \text{ORI}.x \cdot \text{ORI}.z - 2 \cdot \text{ORI}.y \cdot \text{ORI}.scalar) \quad (6.1.5)$$

$$\varphi = \text{atan2}\left(\frac{A}{\cos(\theta)}, \frac{B}{\cos(\theta)}\right) \quad (6.1.6)$$

where:

$$A = 2 \cdot \text{ORI}.y \cdot \text{ORI}.z + 2 \cdot \text{ORI}.x \cdot \text{ORI}.scalar$$

$$B = 1 - 2 \cdot \text{ORI}.x \cdot \text{ORI}.x - 2 \cdot \text{ORI}.y \cdot \text{ORI}.y$$

This algorithm derotates the IMU to the nearest flat level (by the smallest angle). It means that there may occur some undesirable rotation around z -axis during this derotation. It is defined as **zrot** and expressed by (6.1.7). This must be subtracted (the IMU is rotated by the **zrot** quaternion in opposite direction) from computed attitude, see the part of the source code, Figure 6.3.

Now the IMU's tilt is defined by the quaternion **rotToFlat**. Then the last rotation around the z -axis by the heading angle is performed to the orientation of the IMU into the ENU coordinate system. The acceleration in the ENU coordinates is then expressed by the vector **accDerot**. Once the **ORI** is defined, the acceleration **acc** may be rotated by the **ORI** quaternion conjugation. After that, the full size of gravitational acceleration occurs in z -axis and thus the gravitational acceleration (1 g) can be subtracted in the z -axis to getting of the IMU inertial acceleration only.

$$zrot = \text{atan2}\left(\frac{A}{\cos(\theta)}, \frac{B}{\cos(\theta)}\right) \quad (6.1.7)$$

where:

$$A = 2 \cdot \text{ORI}.y \cdot \text{ORI}.x + 2 \cdot \text{ORI}.z \cdot \text{ORI}.scalar$$

$$B = 1 - 2 \cdot \text{ORI}.z \cdot \text{ORI}.z - 2 \cdot \text{ORI}.y \cdot \text{ORI}.y$$


```

// attitude from accelerometer
float alfa, cosAlfa;
float rotVecX, rotVecY, rotVecZ;
float accXlast=0, accYlast=0, accZlast=1;

acc.normalize();
cosAlfa=accXlast*acc.x()+accYlast*acc.y()+accZlast*acc.z();
alfa=acos(cosAlfa);

rotVecX=1/sin(alfa)*((accYlast*acc.z()-accZlast*acc.y()));
rotVecY=1/sin(alfa)*((accZlast*acc.x()-accXlast*acc.z()));
rotVecZ=1/sin(alfa)*((accXlast*acc.y()-accYlast*acc.x()));

// remove parasite
orientation = QQuaternion::fromAxisAndAngle(0,0,-1,zrot)*orientation;
rotToFlat = orientation;
orientation = QQuaternion::fromAxisAndAngle(0,0,1,heading)*orientation;

// derotation
QVector3D accDerot = orientation.rotatedVector(acc);
QVector3D magFlat = rotToFlat.rotatedVector(mag);
    
```

Figure 6.3 A part of c-code for derotation.

As it was written above, when the IMU is still, we can compute the heading from the magnetometer. When the measured data are rotated to flat level (**magFlat**, Figure 6.3) and the result of absolute magnetic field meets the conditions (the outcome has to be found within the interval $\langle 44985.24; 52808.76 \rangle$ nT, this is $\pm 8\%$ from the standard environment where the measurements were performed, obtained as an average value from long-term measurement).

The computation of the heading from magnetometer in 2D is shown in relation (6.1.8); the vector **magFlat** is used. Because of the data measured by magnetometer are transformed into flat level, the determination of the heading (δ) in 2D is correct. In the other case, when the magnetometer data are not derotated into the flat level, another formula for the determination of the declination in 3D has to be used [44].

$$\begin{aligned}
 \delta &= \frac{\pi}{2} - \operatorname{atan}\left(\frac{\operatorname{magFlat}.x}{\operatorname{magFlat}.y}\right) && \text{when } \operatorname{magFlat}.y > 0 \\
 \delta &= 3 \cdot \frac{\pi}{2} - \operatorname{atan}\left(\frac{\operatorname{magFlat}.x}{\operatorname{magFlat}.y}\right) && \text{when } \operatorname{magFlat}.y < 0 \\
 \delta &= \pi && \text{when } \operatorname{magFlat}.y = 0 \quad \text{and} \quad \operatorname{magFlat}.x < 0 \\
 \delta &= 0 && \text{when } \operatorname{magFlat}.y = 0 \quad \text{and} \quad \operatorname{magFlat}.x > 0
 \end{aligned}
 \tag{6.1.8}$$

This gives us the heading (azimuth), the direction of the IMU x -axis due to the magnetic north (inertial x -axis). To be correct, the geodetic declination must be added to get the heading regarding to the geographic north. Of course, it has to be adjusted for different locations by different values, which are determined in map charts [45]. For Brno,

Kohoutovice, the geodetic declination reaches 4.16° [47]. This is positive, east declination and thus the value has to be subtracted from the calculated true azimuth.

6.2 When the state is “walking”

The neural network decides that the IMU state is “walking”. The accelerometer measures the gravitational acceleration mixed with inertial acceleration that it needs to be separated. At first, the transfer from RPY to ENU coordinates is performed. This is quite difficult since the gyroscope has almost full confidence. Then the full size of gravitational acceleration occurs in z -axis again and 1 g can be subtracted to get the inertial acceleration only.

As already mentioned, the accelerometer data cannot be used for the attitude determination and only the data that was measured by the gyroscope have to be use. There is no other way than the application of the mathematical integration, due to there the gradually increasing inaccuracy appears. This is caused by the need of the integration of the rotation rates from the sensor to find out deltas (differences) of particular angles. However the inaccuracy here leads to really huge error while the velocity and position is calculated (another two mathematical integrations of computed accelerations in time).

Figure 6.4 shows how the quaternion **deltaFrame** is defined and how the quaternion **orientation** is rotated by **deltaFrame** to find out the new **orientation**.

```
// attitude from gyro
QQuaternion deltaFrame;
float q0=1, q1=0, q2=0, q3=0, gy, gz, gx;
gx=gyr.x()/180*M_PI;
gy=gyr.y()/180*M_PI;
gz=gyr.z()/180*M_PI;

float qDot1, qDot2, qDot3, qDot4;
qDot1 = 0.5 * (-q1 * gx - q2 * gy - q3 * gz);
qDot2 = 0.5 * (q0 * gx + q2 * gz - q3 * gy);
qDot3 = 0.5 * (q0 * gy - q1 * gz + q3 * gx);
qDot4 = 0.5 * (q0 * gz + q1 * gy - q2 * gx);

q0 += qDot1 * frameTime;
q1 += qDot2 * frameTime;
q2 += qDot3 * frameTime;
q3 += qDot4 * frameTime;
deltaFrame = QQuaternion(q0, q1, q2, q3);

orientation = orientation * deltaFrame; //successive attitude determ.
headG = headingFromQuat(orientation) *180/M_PI; //heading after the mov.
orientation = QQuaternion::fromAxisAndAngle(0, 0, -1, headG)*orientation;
```

Figure 6.4 Single step of successive attitude determination.

The matrix **dcm** (direct cosine matrix, DCM) is created from recalculated quaternion **orientation** (after the **deltaFrame** application), as seen in equation (6.2.1).

$$\begin{aligned}
 dcm(1,1) &= 2 \cdot ORI.scalar^2 - 1 + 2 \cdot ORI.x^2 \\
 dcm(1,2) &= 2 \cdot ORI.x \cdot ORI.y + 2 \cdot ORI.z \cdot ORI.scalar \\
 dcm(1,3) &= 2 \cdot ORI.x \cdot ORI.z - 2 \cdot ORI.y \cdot ORI.scalar \\
 \\
 dcm(2,1) &= 2 \cdot ORI.x \cdot ORI.y - 2 \cdot ORI.z \cdot ORI.scalar \\
 dcm(2,2) &= 2 \cdot ORI.scalar^2 - 1 + 2 \cdot ORI.y^2 & (6.2.1) \\
 dcm(2,3) &= 2 \cdot ORI.y^2 + 2 \cdot ORI.x \cdot ORI.scalar \\
 \\
 dcm(3,1) &= 2 \cdot ORI.x \cdot ORI.z + 2 \cdot ORI.y \cdot ORI.scalar \\
 dcm(3,2) &= 2 \cdot ORI.y^2 - 2 \cdot ORI.x \cdot ORI.scalar \\
 dcm(3,3) &= 2 \cdot ORI.scalar^2 - 1 + 2 \cdot ORI.z^2
 \end{aligned}$$

All over, the angles θ and φ are determined from quaternion by formulas (6.1.5) and (6.1.6) where is applied:

$$\begin{aligned}
 A &= 2 \cdot ORI.x \cdot ORI.z + 2 \cdot ORI.y \cdot ORI.scalar & (6.2.2) \\
 B &= 1 - 2 \cdot ORI.x^2 - 2 \cdot ORI.y^2
 \end{aligned}$$

The rotation around the z -axis, **zrotation**, is computed by equation (6.2.3). The rotation around the z -axis is then performed. The acceleration in the ENU coordinate frame is then expressed by the **accDerot** vector.

$$zrotation = atan2\left(\frac{A}{\cos(\theta)}, \frac{B}{\cos(\theta)}\right) \quad (6.2.3)$$

where:

$$\begin{aligned}
 A &= 2 \cdot ORI.y \cdot ORI.x + 2 \cdot ORI.z \cdot ORI.scalar \\
 B &= 1 - 2 \cdot ORI.z^2 - 2 \cdot ORI.y^2
 \end{aligned}$$

The same result of the IMU orientation from the DCM can be obtained while the vector multiplication of the **dcm** with measured acceleration vector is used. Nevertheless, the problem with the gimbal lock effect persists.

The heading is computed by the same way as in the case when the IMU is “still”. In this case, when the IMU state is “walking” there is not parasitic rotation present. On the other hand, the heading is computed only from the integrated data (that was measured by the gyroscope) as well as the tilt and this leads to growing inaccuracy in time, as mentioned above. We can improve the accuracy by replacing heading calculated from the gyroscope by absolute heading from the magnetometer.

6.3 Problems

Because of data history is needed in process, the type of the ANN cannot be from group of the classification ANNs. It results in a fact that the continuous output is in the range from -1 to +1 (in classification ANN the output is exactly -1 or +1, see Chapter 5.4). To distinguish of two states (walking and staying still) a decision border has to be defined.

Also, the neural network output is filtered by a KF and due to this filtering, some delay in network decision on state occurs. The movement is then evaluated later and the orientation is determined in a bad way.

To avoid this, two additional steps are performed:

1. Auxiliary condition has to be met to classify the IMU state as “still”, see (6.3.1), where δ_{acc} is the actual deviation in [g] of the measured acceleration and (6.3.2) where δ_{gyr} is defined by maximum value of rotation rate. This condition block is called **SillyStatus** filter and it returns “0” when any of the values was out of the required range in the last N samples, otherwise it returns “1”. Tolerance of acceleration deviation δ_{ACC} was set to 0.01 g and the rotation rate tolerance was set to 2 °/s. Figure 7.3 shows the **Tracker setting dialog** for these values.

$$\delta_{acc} = \sqrt{(acc_x^2 + acc_y^2 + acc_z^2)} - 1 \quad (6.3.1)$$

$$\delta_{gyr} = \max(gyr_x, gyr_y, gyr_z) \quad (6.3.2)$$

2. The second step is to delay the data processing by N samples in order to be able to “see the future”. Whenever the state changes to “walking”, the walk processing algorithm runs using N samples in advance and thus the KF delay and ANN delay are eliminated. This ensures that the **orientation** quaternion is not absolutely computed from accelerometer while the IMU is already moving.

Another issue is heading computation when the IMU’s x-axis points upwards or downwards. When such a situation occurs, the heading is not defined and the **orientation** (particularly the rotation around ENU z-axis) is computed fully from rotation rate sensor, regardless of whether the IMU is “still” or not.

6.4 Filtering

In my thesis, Kalman filter is used in several cases. At first, Kalman filter affects the output of the artificial neural network; the other KF is used for the rotation rate and acceleration evaluation. All of them use the linearization of the system model.

It is also suitable to filtering of the measured magnetic field. The short-term deviations are smoothened and the heading determination is then more stable. We do not apply the filter on the determined heading values since the heading values are always found in $\langle -\pi; \pi \rangle$ interval.

6.4.1 KF applied on ANN outputs

The maximum deviations of the ANN outputs (z_k) from optimum output values (-1, +1) typically reach up to approx. 0.5 and here the Kalman filtering seems to be very desirable. The Kalman gain is computed based on the theory according to (6.4.1). Then the a priori state estimation is performed (6.4.2) and the a posteriori error covariance matrix estimation (a measure of the estimated accuracy of the state estimate) is

corrected (6.4.3). The predicted value of estimated state vector (6.4.4) and the predicted value of estimation covariance (6.4.5) is computed.

$$\mathbf{S} = \mathbf{H}\mathbf{P}_k^-\mathbf{H}^T + \mathbf{R}$$

$$\mathbf{K}_k = \frac{\mathbf{P}_k^-\mathbf{H}^T}{\mathbf{S}} = \frac{\mathbf{P}_k^-\mathbf{H}^T}{\mathbf{H}\mathbf{P}_k^-\mathbf{H}^T + \mathbf{R}} \quad (6.4.1)$$

$$\mathbf{x}_k^- = \mathbf{A}\mathbf{x}_{k-1} \quad (6.4.2)$$

$$\mathbf{P}_k^- = \mathbf{A}_k\mathbf{P}_{k-1}\mathbf{A}_k^T + \mathbf{Q}_{k-1} \quad (6.4.3)$$

$$\mathbf{x}_k = \mathbf{x}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}(\mathbf{x}_k^-, 0)) \quad (6.4.4)$$

$$\mathbf{P}_k = \mathbf{P}_k^- - \mathbf{K}_k\mathbf{H}\mathbf{P}_k^- = (\mathbf{1} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^- \quad (6.4.5)$$

Because of the course of the function (continuous ANN output) is relatively simple, linear Kalman filter is used and its entities \mathbf{A} (the state-transition model), \mathbf{H} (the observation model), \mathbf{P}_0 (input a priori estimate covariance), \mathbf{P}_k (a priori estimate covariance), \mathbf{Q} (the covariance of the process noise) and \mathbf{R} (the covariance of the observation noise) are set experimentally in order (6.4.6):

$$\begin{aligned} \mathbf{A} &= [1] \\ \mathbf{H} &= [1] \\ \mathbf{Q} &= [0.5] \\ \mathbf{P}_0 &= [0.1] \\ \mathbf{P}_k &= \mathbf{F}\mathbf{P}_{(k-1)}\mathbf{F}^T + \mathbf{Q} \\ \mathbf{R} &= [5] \end{aligned} \quad (6.4.6)$$

Sensitivity matrix \mathbf{H} was set to one, thus (6.4.7) applies. Corrected value of estimated state vector is then computed according to (6.4.8). The predicted measurement value is based on weighted arithmetic mean of previous state vectors summed with integrated value of measurement. The error covariance is updated (6.4.9), based on (6.4.5), and the algorithm repeats.

$$\mathbf{K}_k = \frac{\mathbf{P}_k^-}{\mathbf{P}_k^- + \mathbf{R}} \quad (6.4.7)$$

$$\mathbf{x}_k = \mathbf{x}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{x}_k^-) \quad (6.4.8)$$

$$\mathbf{P}_k = (\mathbf{1} - \mathbf{K}_k)\mathbf{P}_k^- \quad (6.4.9)$$

Thus, the output of the ANN in this work is filtered by the LKF to avoid oscillations around the decision boundary which eliminates the hopping between “+1” and “-1” status of the IMU. In addition, because of the output of the ANN is not exactly “+1” or “-1”, the filter modifies the raw ANN outputs to get closer to expected values. By this

way the LKF output is prepared for the hard limit filter application. The improvement of the ANN output values after the LKF in time is shown in Figure 6.5.

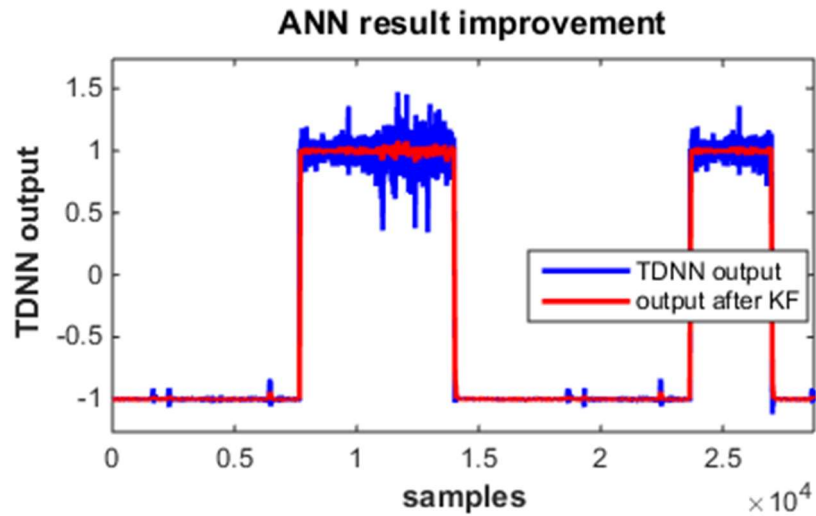


Figure 6.5 Improved output of the artificial neural network (filtered by LKF).

6.4.2 Kalman filtering of sensor data

Because of the quaternion **orientation** is computed only from the accelerometer data when the state is “still”, the acceleration data may not be filtered before the **orientation** determination and hence before the transformation from the RPY coordinates to ENU coordinates. After that derotation, the acceleration data should be filtered by LKF and used for further evaluation of the velocity and consequently of the position (Figure 6.6).

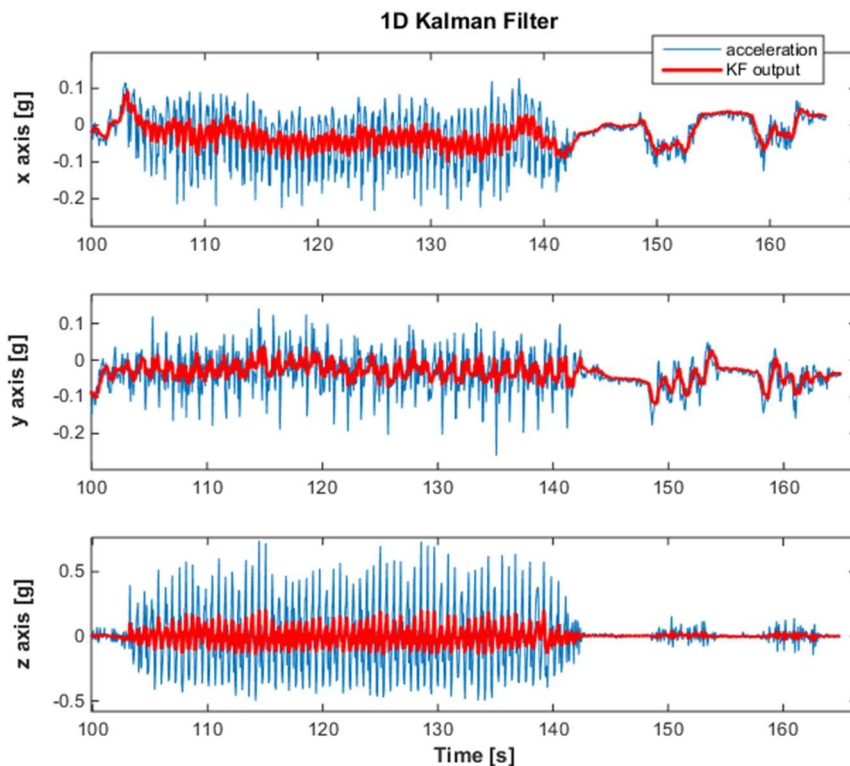


Figure 6.6 Kalman filter applied on the acceleration data.

The quaternion **orientation** is computed from the rotation rate sensor when the state is “walk”, but that sensor is not used when the state is “still”. Thus, the filtration of the rotation rates must be performed very sensitively. The filter contains two groups of coefficients; the first of them is used when the state is “still” and the second one when the state is “walk”. Then the deviations are suppressed highly when the IMU remains stationary and they are satisfactorily filtered when the IMU “walks”. The situation is clear from Figure 6.7.

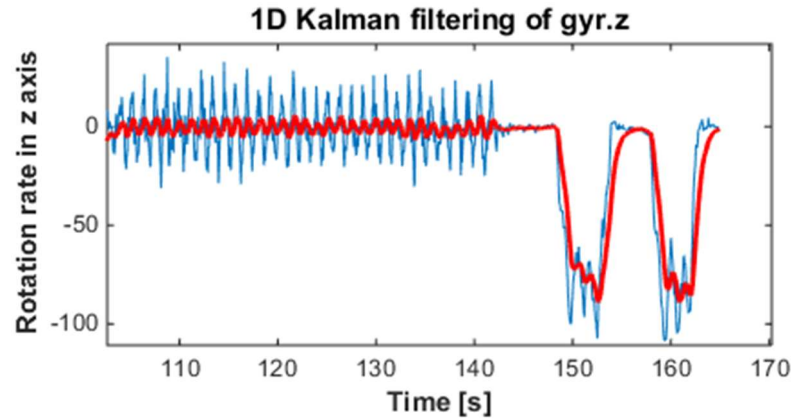


Figure 6.7 Kalman filter applied on the rotation rate sensor data.

The data from magnetometer is also filtered to ensure as high smoothness of the particular magnetic strengths in time as possible. The filtration of the heading is inconvenient due to the interval of values ($-\pi; \pi$) or $\langle 0; 2\pi \rangle$ depending on relation for the heading determination. It ensures a smoother graph of the heading in time, in other words, the direction of found North does not vary in short-term conditions from the IMU’s RPY coordinates point of view.

6.4.3 Kalman filtering of velocity and position

The filtration of the velocity and position is not necessary because the derotated accelerometer data are already filtered and thus any further filtering even after integration is redundant and undesirable.

6.4.4 Additional filtration methods

Moving Average Filter

In principal, a new value is averaged with given count of previous values k and resulting value is used. Then, the smoother curve of signal is obtained. The size of the filtration window indicates the degree of smoothness; with higher k the line is smoother, however, the resulting time-delay rises.

Simple Low Pass Filter

The smoother curve is in this case obtained as well. The weight of a new sample relative to the previous is determined and the result value is affected by the weighted previous sample. The implementation has various forms, see (6.4.10) - (6.4.11):

$$x_{result} = \alpha \cdot x_{k-1} + (1 - \alpha) \cdot x_k \quad (6.4.10)$$

$$x_{result} = x_{k-1} + (1 - \alpha) \cdot (x_k - x_{k-1}) \quad (6.4.11)$$

$$x_{result} = x_{k-1} + \frac{(x_k - x_{k-1})}{\alpha} \tag{6.4.12}$$

The smoothness of resulting curve depends on α , in case of multiplication $\alpha \in \langle 0;1 \rangle$, in case of division $\alpha \in \langle 1;\infty \rangle$. This filtering brings the same disadvantage, with higher α the time-delay rises.

Thresholding

The output values are compared with a pre-set threshold value. In case of hard thresholding the measured value is set to zero when the output is less than the pre-set threshold value. In case of the soft thresholding, the pre-set threshold value is subtracted from every sample. That means the signal is shifted by the pre-set threshold value.

When data from accelerometer is processed, the thresholding might not be used. Nevertheless, this filtration is suitable in the soft calibration process.

6.4.5 Additional filters implementation

Gyroscope raw output

The data from gyroscope was received for 30 minutes when the IMU laid on a table in a room. Figure 6.8 shows particular rotations in time. The bias offset (the mean value) and standard deviation along particular axes are shown in TABLE 6.1.

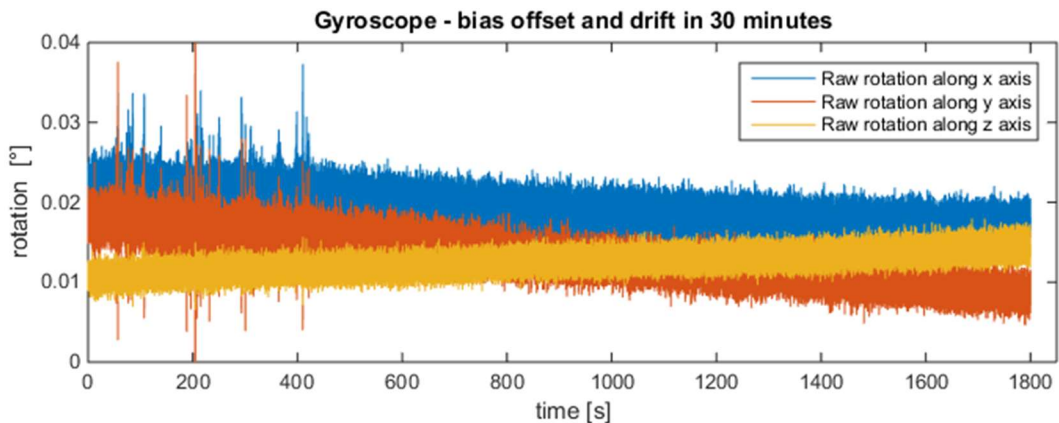


Figure 6.8 Raw particular rotations in time when IMU is still – 30 minutes.

From the measured data we can deduce that the offset error (the shift of zero level) is quite small and it may be filtered by the moving average filter when the IMU is still. The dynamic error that is caused by a signal fluctuation (drift) affects the result orientation significantly. The tilt angle increases due to the drift though the IMU is still, and this effect, in this case, deeply applies when the IMU is moving and the orientation of the IMU is computed from the rotation rates.

TABLE 6.1 Bias offset and standard deviation of raw rotations when the IMU is still.

	x-axis	y-axis	z-axis
bias offset [°]	$1.910 \cdot 10^{-2}$	$1.311 \cdot 10^{-2}$	$1.251 \cdot 10^{-2}$
standard deviation [°]	$2.198 \cdot 10^{-3}$	$3.105 \cdot 10^{-3}$	$1.361 \cdot 10^{-3}$

Thus the low-pass filter according to (6.4.10) formula was chosen to filter the rotation rates during the walk with smaller alpha coefficient – it brings quite small time delay

after the filtration. To be accurate, the thresholding is used to decide whether the IMU is still or not when the UOG (use only gyro) mode is turned on. Then, α differs due to defined state of the IMU. When the IMU is still, alpha is set to almost one. The time delay then causes negligible errors.

Acceleration raw output

The measured acceleration (after the hard offset compensation and axes scaling) achieves the absolute value of 1 g (the gravitational acceleration) while no other forces (movements) are present. When the IMU is turned on, the necessary condition for the correct measurement is that the IMU has to stay still at least for 2 seconds. Because of the motion brings almost always attitude changes, there is no way how to filter the acceleration sensor drift in time.

Despite of that, this system allows to estimate the velocity and position based on the artificial neural network output and whenever the IMU state is „still“ the velocity is set to zero and thus the position is not changing. Then drifts do not apply. The impact of the accelerometer drift is shown in Figure 6.9. The error rises significantly during the time due to the integrations (actual velocity and position is computed based on equations (6.4.1) - (6.4.3)). In our system we suppose that the state changes frequently (the figure shows 23 minutes of the measurement while the IMU was laying on a table).

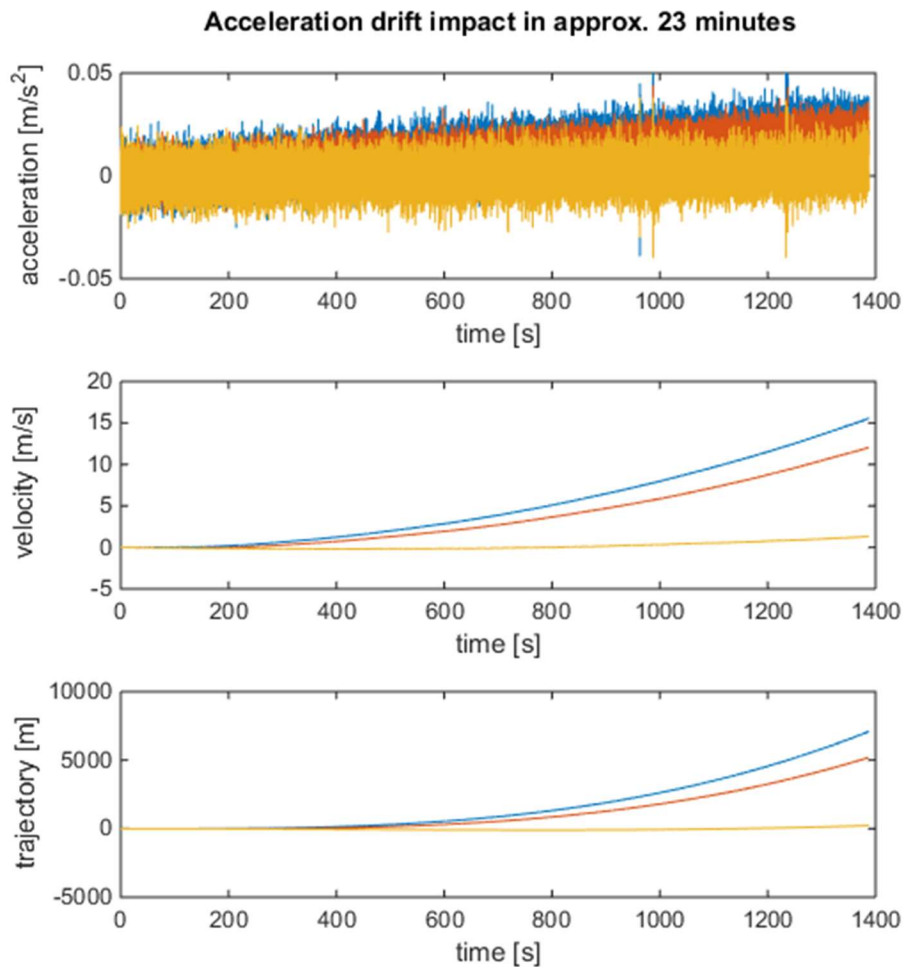


Figure 6.9 Acceleration drift impact.

7 STRAPDOWN NAVIGATION IMPLEMENTATION

Once the IMU coordinate system is transformed into the ENU coordinates, the inertial velocity and position of the device can be computed.

7.1 Basics of strapdown algorithm

The basic formulas of strapdown algorithm are shown in relations (7.1.1), (7.1.2) and (7.1.3). Since the acceleration is measured discontinuously, we use deltas in equations to express the velocity and the trajectory in time.

$$\bar{v}_a = \frac{v_{(k-1)} + v_k}{2} \quad (7.1.1)$$

$$v_k = v_{(k-1)} + acc \cdot \Delta t \quad (7.1.2)$$

$$s_k = s_{(k-1)} + v_k \cdot \Delta t + \frac{1}{2} \cdot a \cdot \Delta t^2 \quad (7.1.3)$$

A very important correction step now is to set the derotated acceleration vector to (0, 0, 1) in ENU coordinates and inertial velocity vector to (0, 0, 0) when the state is “still”. If we do not process it we obtain an acceleration value very close to zero, but the velocity value would change more significantly. During the “still” period, the position of the IMU would change, which is unacceptable, moreover when the state changes to “walking”, evaluation of velocity and position would be affected by this inaccuracy. Next part describes the developed software for strapdown navigation. Once the application receives data from the IMU, the steps described further may be performed.

7.2 Software for Inertial Measurement Unit – Tracker

As a suitable environment Qt has been chosen. Qt is the leading independent technology for cross-platform development. An application for data processing and visualization was created including visualization. Besides the main.cpp file, further C++ source files have been developed:

Main source codes for processing:

receiver.cpp
 calibrator.cpp
 derotator.cpp
 integrator.cpp
 filehandler.cpp
 brain.cpp

ANN source codes:

genericnetwork.cpp
 mod.cpp

Utils source codes:

brain.cpp

pplotgroup.cpp
 putils.cpp

User interface source codes:

cubedialog.cpp
 customtab.cpp
 packformater.cpp
 settings.cpp
 vizualizer.cpp

Visuals source codes:

calstatus.cpp
 glcubevidget.cpp

qcustomplot.cpp
tcompass.cpp

Next Chapters contain a description of main source files of the application.

7.2.1 Receiver

When data is sent from Arduino UNO and serial link is opened the data is available for processing in Qt. There are two possible ways how to transmit the data, the first is via USB cable and the second is wireless, via Bluetooth. Received data are converted to the appropriate physical units, depending on the sensitivity (how the output registers were set, see page 102, APPENDIX C), as further lines show.

```
//Sensor Units Conversion Constants  
  
magLSB_Gaus = 1/6.842e3;    //+- 4 gauss  
gyrMDPS_LSB = 17.5e-3;    //+- 500 °/s  
accMg_LSB = 0.061e-3;    //+- 2 g
```

```
//Captured data conversion (to appropriate units)  
  
//Conversion of accelerometer data to acceleration in [g]  
  
acc.setX(rawData[6]*accMg_LSB);  
acc.setY(rawData[7]*accMg_LSB);  
acc.setZ(rawData[8]*accMg_LSB);  
  
// Conversion of gyro data to rotation rate in [deg/s]  
  
gyr.setX(rawData[3]*gyrMDPS_LSB);  
gyr.setY(rawData[4]*gyrMDPS_LSB);  
gyr.setZ(rawData[5]*gyrMDPS_LSB);  
  
// Conversion of magnetometer data to magnetic field in [gauss]  
  
// x-axis is represented by y-axis  
// y-axis is represented by x-axis in the opposite direction  
  
mag.setX(rawData[1]*magLSB_Gaus);  
mag.setY(-rawData[0]*magLSB_Gaus);  
mag.setZ(rawData[2]*magLSB_Gaus);
```

To align the coordinate frames of sensors, magnetometer coordinate axes had to be swapped – this is clear from the last part of c-code above. The offsets and the scale factors of sensor axes were computed from experimentally found out (measured) data.

In addition, the expected measured acceleration is 1 g in the appropriate axis while the IMU is still and the axis points exactly downwards. In practise the value is little bit different from the 1 g and it differs for each of the axes as shows the c-code part below. Those differences have to be compensated and the measured values are thus shifted and scaled.

```

//Hardware constants of accelerometer

accMin = QVector3D(-1.0055485, -0.9946478, -0.9949894);
accMax = QVector3D( 1.0008940,  1.0017908,  1.0239429);

//Offset

acc.setX(acc.x() - ((accMax.x() + accMin.x()) / 2));
acc.setY(acc.y() - ((accMax.y() + accMin.y()) / 2));
acc.setZ(acc.z() - ((accMax.z() + accMin.z()) / 2));

//Scale

acc.setX(acc.x() / ((accMax.x() - accMin.x()) / 2));
acc.setY(acc.y() / ((accMax.y() - accMin.y()) / 2));
acc.setZ(acc.z() / ((accMax.z() - accMin.z()) / 2));
    
```

Figure 7.1 shows the converted raw data – acceleration in [g], rotation rate in [deg/s] and the magnetic strength in [gauss]. The blue colour represents the data measured in x-axis, the green colour represents the data measured in y-axis and the red one represents the data measured in z-axis.

Numbers on the right side represent last measured value in appropriate axis. The motion that is captured in this figure represents walking. As you can see from the depicted accelerations and rotation rates, four steps in approximately same direction were performed during this experiment.

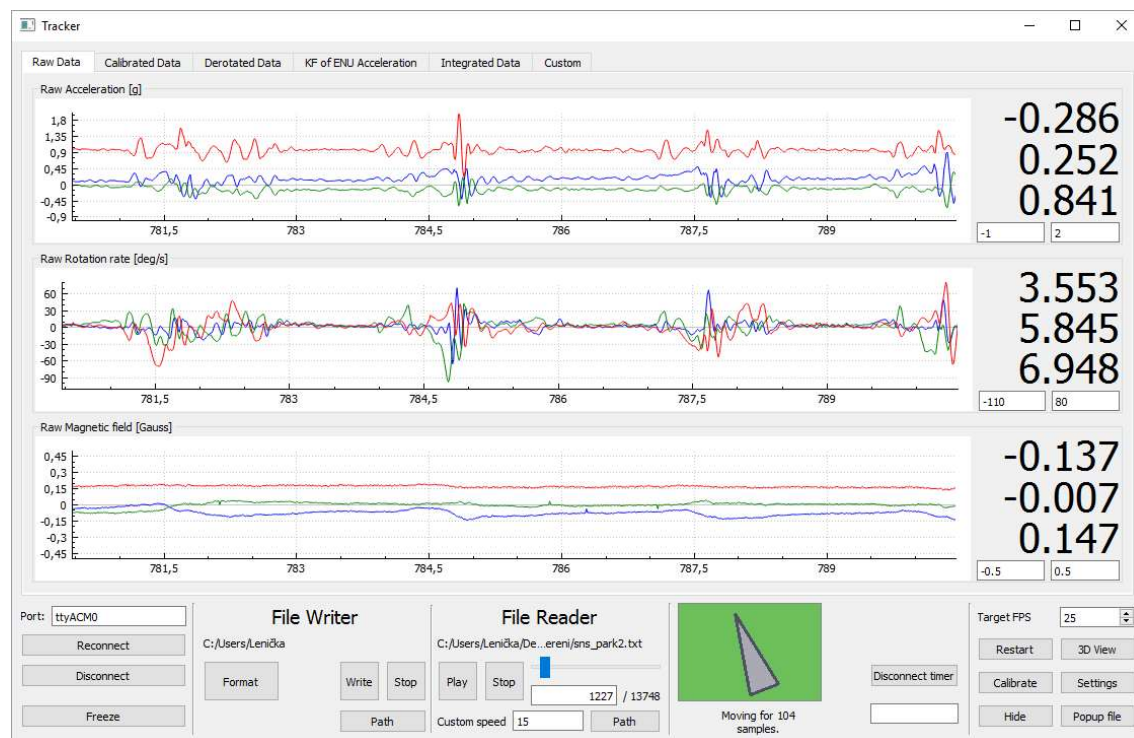


Figure 7.1 Raw data.

7.2.2 Calibrator

This part calibrates received data for the environment where the measurement is performed. The adjustable values for magnetometer calibration are local magnetic strength, linear Kalman filter constants for magnetometer data filtration and the length of history (the previous sensor data that are available in each time-step). The process of active calibration takes about two seconds.

```

//Mag offset
mag.setX(mag.x() - ((magMax.x() + magMin.x()) / 2));
mag.setY(mag.y() - ((magMax.y() + magMin.y()) / 2));
mag.setZ(mag.z() - ((magMax.z() + magMin.z()) / 2));

//Mag scale
mag.setX(mag.x() / ((magMax.x() - magMin.x()) / 2));
mag.setY(mag.y() / ((magMax.y() - magMin.y()) / 2));
mag.setZ(mag.z() / ((magMax.z() - magMin.z()) / 2));

//Magnetic local field strength
mag = MagLocal * mag;

//KF for mag
mag = kalMag->getrValue(mag);
    
```

Always when the IMU is “still”, the gyro data offset is found out and this offset is subtracted (passive calibration). The subtraction of the last detected offset value is applied also during the “walking”. We assume that no movement occurs when the calibration is running, thus the acceleration and the magnetic field may be averaged to get the first **orientation** quaternion. In addition, the mode (*state*) is defined at the end of the calibration.

Figure 7.1 and Figure 7.2 show the same data sample, before and after the calibration. The significant difference can be found in the graph for the magnetic strength. Now the values are converted into [mGauss] and the total magnetic field corresponds to the place where the measurement was performed (see APPENDIX B).

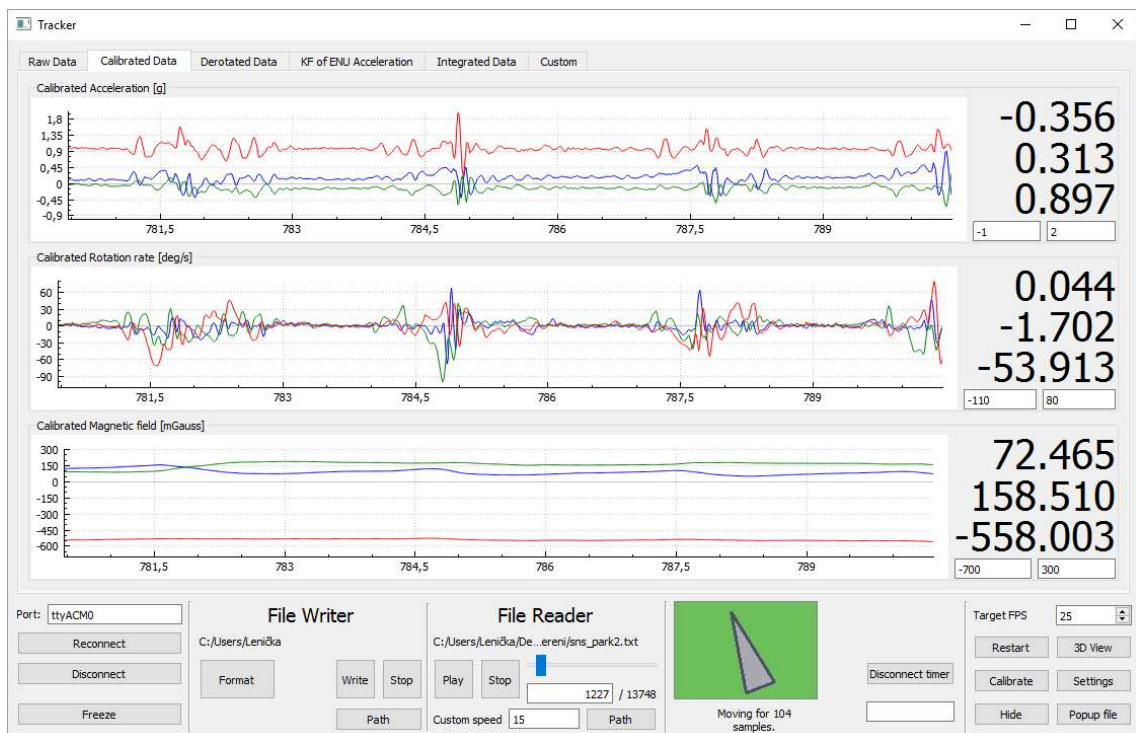


Figure 7.2 Calibrated data.

The active calibration may be also called during the measurement by the user. It requires keeping the IMU still for at least two seconds. The calibration resets the

position to (0, 0, 0), resets the velocity to (0, 0, 0) and at the end of the calibration the **orientation** quaternion is recomputed based on the averaged accelerations.

7.2.3 Derotator

The derotation algorithms were described in Chapter 0. Based on the settings, in addition, the derotator may use “*Use Only Gyro*” (**UOG**) mode or **SillyStatus** mode. When the **UOG** mode is activated, the measured acceleration vector is derotated into ENU coordinate system by the quaternion deltas that are fully defined by the data from the calibrated and compensated rotation rate sensor. The data processing is then similar as in Chapter 6.2: When the state is “walking”.

The **SillyStatus** mode works as a hardlimit filter and it defines the state of the IMU based only on the filter output. Providing the data deviations from reference are lower than user-defined decision values, the state of the IMU is “still”. In that case, we expect that the absolute acceleration equals 1 g (the deflection is set in [g]). We also expect that the IMU does not rotate, thus the expected value from gyroscope is zero in each axis (the deflection is set in degrees). This situation may occur also incidentally, thus the still limit defines the minimal number of the successive samples that match given conditions, before the state is changed to “still”. The tolerance of acceleration deflection (“*Acc limit*”), the tolerance of rotation rate deflection (“*Gyro limit*”) and (“*Still limit*”) for SillyStatus mode are set in the setting dialog window.

There are several artificial neural networks that may decide on the actual state of the IMU. Thus the “*Select NN*” allows to define the ANN that is used. The activation of the ANN processing is given by checking the “*Neural network*” checkbox. Constants for ANN Kalman filter (“*KF for NN*”) and “*NN status boundary*” can be set by the user before or during the measurement in the setting dialog window. The boundary value defines the border between the final “still” decision and final “walking” decision. The ANN output after the Kalman filtration is compared to the given value. We expect the ANN output values between -1 and 1. We can also use the combination of the SillyStatus mode and ANN implementation. The ANN then defines the “still” and “walking” phases. Nevertheless, the quaternion orientation is recomputed and corrected only in case that the SillyStatus filter decides that the IMU is “still” and the ANN detects the “still” state too.

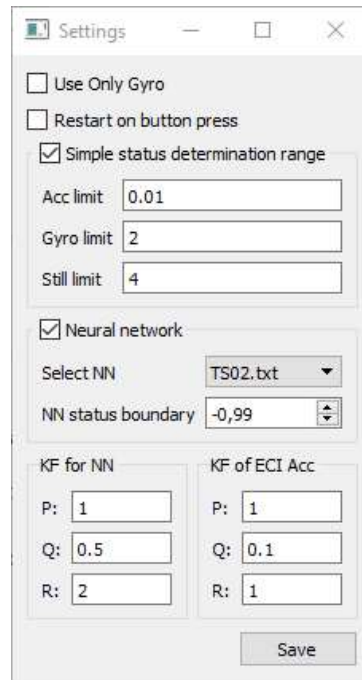


Figure 7.3 Setting dialog window for Tracker.

The heading value is computed from the magnetometer regardless of the defined state. In cases when the heading cannot be determined from the magnetometer neither from the gyroscope (the x -axis points upwards or downwards or the tilt is in given range from those directions), the heading is not further computed and the **orientation** quaternion is updated only by the **deltaFrame** quaternion using the gyroscope.

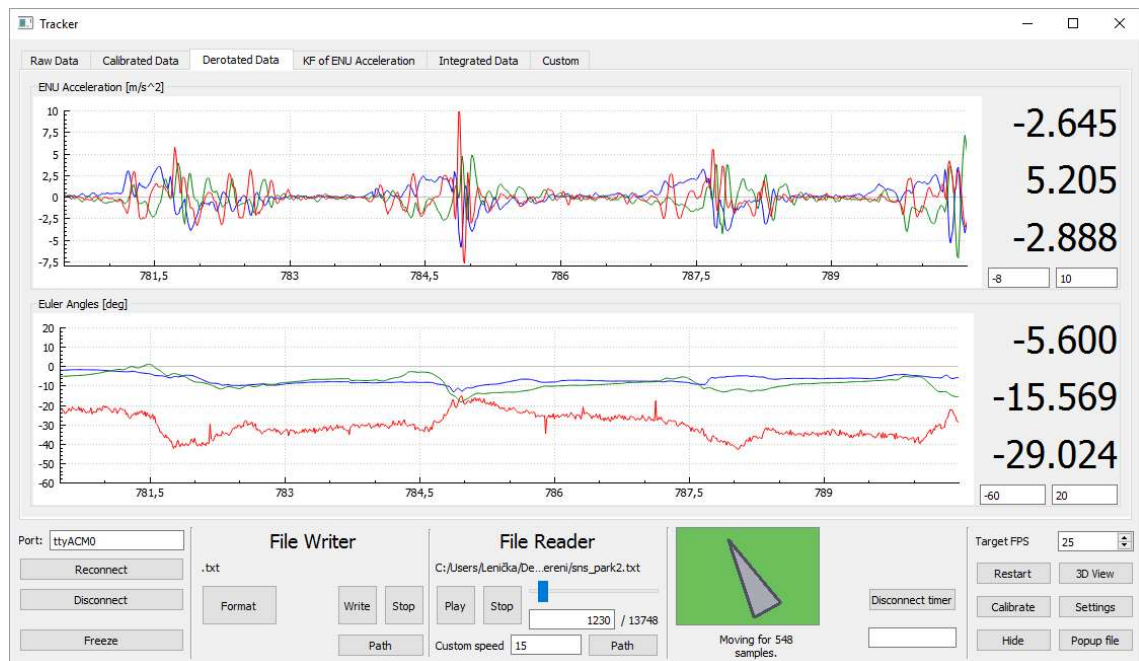


Figure 7.4 Derotated data.

Figure 7.4 shows the data of the same measurement (see Figure 7.1 and Figure 7.2) after the derotation. The acceleration presented in the figure is in ENU coordinate system and because of the subtraction of gravitational force from the z -axis, the acceleration is “fully linear”. It is also converted from $[g]$ to $[ms^{-2}]$.

The bottom graph shows evaluated Euler angles (φ , θ , ψ) of the IMU in the ENU coordinates. The red curve represents the ψ angle, which is equivalent to the heading. All angles are defined in $(-180; 180)$ interval range in [deg]. The Kalman filtration of magnetic strength data has an impact on the evaluated heading value. This graph shows the situation when the magnetic strength was not filtered by a KF. Thus there are cases when the heading value hops by few degrees, e.g. at time about 782 s, 786 s and 787 s.

7.2.4 Integrator

The integrator evaluates the final output values, the velocity and position in time. It operates depending on the state; appropriate computations are further performed. The consequence of defined mode in each possible state is clearly shown in TABLE 7.1. When the heading value from magnetometer cannot be determined, the velocity and position is computed by the same way as in UOG mode.

TABLE 7.1 *Consequences of defined state in particular modes.*

mode	state	consequence
UOG	ANY	orientation of the IMU is defined by the calibrated gyroscope data only
SillyStatus	STILL	orientation of the IMU is recalculated from the accelerometer data only
	OTHER	the UOG mode data processing applies
ANN	STILL	global acceleration is set to (0, 0, 1), actual velocity is set to (0, 0, 0)
	WALK	the UOG mode data processing applies
ANN&SillyStatus	STILL	orientation of the IMU is recalculated from the accelerometer data only global acceleration is set to (0, 0, 1), actual velocity is set to (0, 0, 0)
	WALK	the UOG mode data processing applies

Then, (6.4.2) and (6.4.3) formulas are used to get the velocity and position in inertial 3D ENU coordinate frame. In the presented experiment four human steps can be clearly recognized, see Figure 7.5. They may be detected from the velocity curves or from the position defined below. Between steps, the acceleration, respectively velocity, drops to zero. Of course, during human steps the algorithm thoroughly evaluates the measured data.

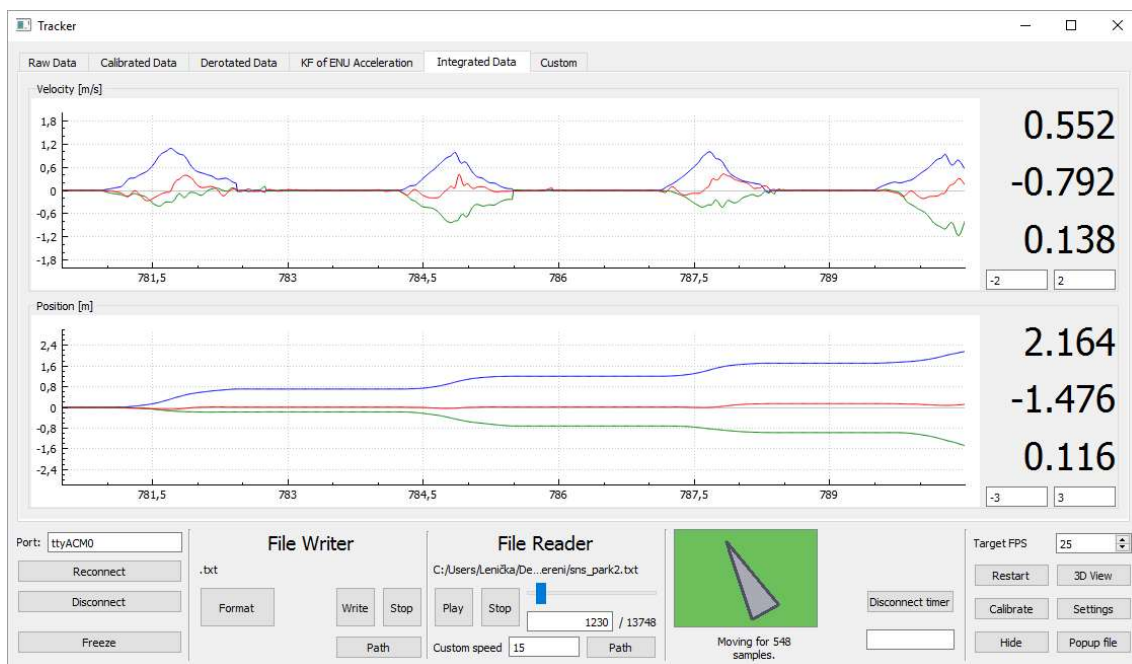


Figure 7.5 Integrated data.

7.2.5 Custom graphs

The software allows to set a number of the custom plots in which the graphs of chosen variables are shown in one window. For example, as Figure 7.6 shows, the calibrated acceleration and artificial neural network output. In the “State and Neural Network output” window you see three curves. The red one represents the SillyStatus filter output (“still” state is defined when the output value is “1”). The green one represents the ANN output and the blue one the ANN output after KF. In those ANN outputs, “still” state is defined when the output value is lower than the “NN status boundary” value.



Figure 7.6 Example of Custom plot data

The next table shows all values that may be plotted in the application.

TABLE 7.2 Variables available in custom plot view.

Raw Acc.	Raw accelerometer data	Cal. Acc.	Calibrated accelerometer data
Raw Gyro	Raw gyroscope data	Cal. Gyro	Calibrated gyroscope data
Raw Mag.	Raw magnetometer data	Cal. Mag.	Calibrated magnetometer data
ENU ACC	Linear acceleration in ENU coordinate frame	ENU KF ACC	Linear acceleration in ENU coordinate frame after KF
Euler angles	Euler angles of actual orientation	Neural output	ANN output, ANN output after KF, combined state (with SillyStatus)
Velocity	Velocity in ENU coordinate frame	Position	Position in ENU coordinate frame

7.2.6 FileHandler

As it is indicated in the name, this part allows to record data to plain text format and also load the data from such a format. The user may configure data format to be saved. It seems to be suitable to choose the set that is measured by IMU (raw rotation rates, raw accelerations, raw magnetometer, and time) because of the further data processing (e.g. possibility of calibration) in Tracker software. The list of all variables (output variables) that can be saved are shown in Figure 7.7. The linear acceleration in ENU coordinates (ENU acc) may be further processed in MATLAB™, for example. The “Disconnect timer” defines the period [ms]; when the timer is switched on, after expiration of the pre-set period the connection with the IMU is closed.

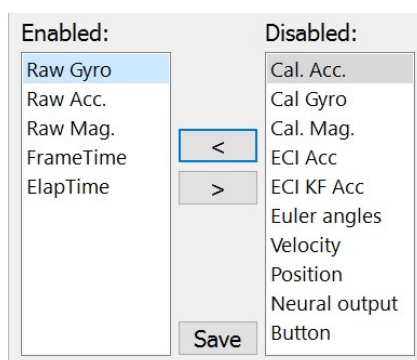


Figure 7.7 A dialog window for recording and replaying the data.

When the saved data is red, the custom speed of reading may be set. The reading may be paused or stopped.

7.2.7 Graphical representations in software

In the bottom of the application window, there are labels and buttons that allow the control of the software (Figure 7.8). At first, the port can be set in case we process the IMU data real-time. There is the possibility to reconnect and disconnect the IMU and also to freeze the view. The File Writer and File Reader allow to define the read/write data format. The path may be also defined by the user. The custom speed defines the speed of data flow in graphs. The menu on the right allows to restart the evaluation process, load the calibration process, hide graphs during the measurement, show the

setting dialog window, pop-up the file writer and file reader to a separate dialog window and show the 3D Cube View window.

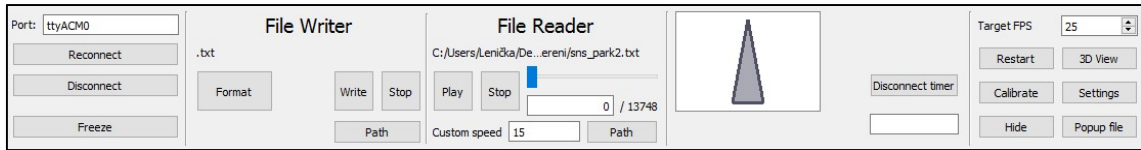


Figure 7.8 The software controls.

Information about the heading is represented by an arrow in the rectangle that points upwards while the IMU (body frame) *x*-axis points to the north. In addition, the information about state is represented by the green/yellow/red background of the rectangle. The number of samples corresponding to the last defined state is shown under this rectangle. The visual output is shown in Figure 7.9.

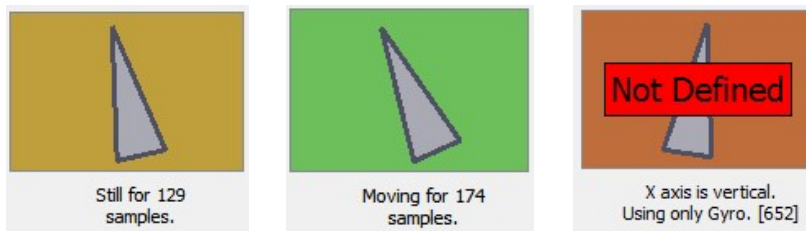


Figure 7.9 Visual information about the heading and the state

The next visual result is the cube in 3D that represents the IMU body (see Figure 7.10). The window that allows to track the IMU also contains information about the number of samples per second (SPS), current position and velocity, and the actual orientation quaternion value. The user may set the view zoom or lock the cube on (0, 0, 0) position (the centre of the screen). Then only the orientation of the cube in ENU coordinates is shown in time. The user may also call the calibration or reset the view (zoom and viewport adjustments) from this screen.

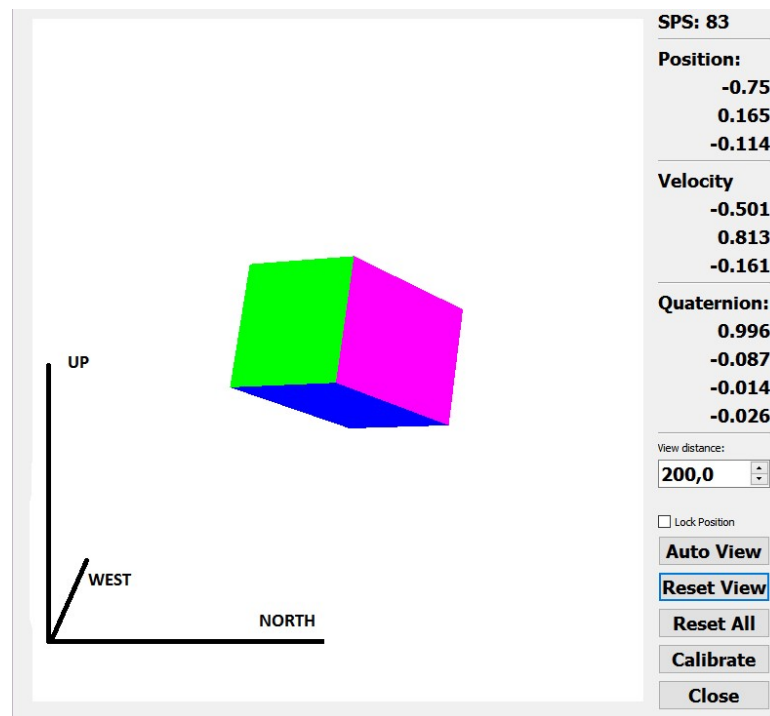


Figure 7.10 Cube view

8 VERIFICATION BY MEASUREMENT

To verify achieved improvements, five situations were performed. The first two measurements contain data when the IMU stays still on the table and in the hand. Next measurement contains data when the IMU is held in the hand and the still phase alternates with step phase. This is called Step and Stop. Last two measurement contain data when the IMU is held in the hand in the still phase to calibration for three seconds and follows with discontinuous or fluent walking.

Results are presented in this way:

The first two measurements were performed while the IMU stood still. The expected ANN output is thus always “still” and the results of the inertial accelerations in ENU coordinates or the ANN outputs are not presented. For comparison, the velocity and the position for each measurement was evaluated by three methods:

- evaluation based only on the calibrated and compensated inertial sensor data (UOG mode)
- evaluation based on the SillyStatus filter that decides whether the IMU is still or not
- evaluation based on the ANN decision whether the IMU is still or walking supported by SillyStatus

The results of the measurements containing any walking are also evaluated by these three methods. The ANN output and results using SillyStatus mode are shown in graphs. Furthermore, the velocity and position in ENU coordinate system is depicted.

8.1 Case I: IMU stayed still on the table

The velocity and the position in time is shown in tables below. The values were taken after 5, 10, 30, 60 and 120 seconds of the measurement. The initial velocity is $v = (0, 0, 0) \text{ ms}^{-1}$ and initial position is $s = (0, 0, 0) \text{ m}$. TABLE 8.1 shows the results when the UOG mode was activated. TABLE 8.2 shows the results when the SillyStatus filter was used only and TABLE 8.3 shows the results with application of ANN with SillyStatus. Because of the zero values of the velocity and thus position in time, the absolute values are presented in the third table only. From these results the fact that SillyStatus filter works properly is clear. The UOG mode results verify that the inertial sensors were correctly calibrated. The ANN operates with 100 % of reliability.

Figure 8.1 shows the UOG mode curve depicted by the red colour, the SillyStatus mode curve depicted by the blue colour and the ANN aided result curve depicted by the green one. The result of the UOG mode is in meters, the result of the SillyStatus mode and ANN mode are in millimetres (the second axis on the right hand side). The setting dialog window in Figure 7.3 shows the values of the parameters that were used while the data was processed in SillyStatus mode and which parameters were set while the data was processed with the ANN:

- The “Acc limit” value was set to 0.01.
- The “Gyro limit” value was set to 2.
- The “still limit” value was set to 4.

- The “NN status boundary” value was set to -0.99.
- KF for ANN constants: P = 1; Q = 0.5; R = 2.

TABLE 8.1 *Velocity and position, UOG mode still on the table.*

Time [s]	5	10	30	60	120
v in x-axis [m/s]	-0.012	-0.077	-1.504	-5.812	-24.846
v in y-axis [m/s]	-0.005	-0.083	-2.259	-10.911	-48.498
v in z-axis [m/s]	-0.147	-0.351	-1.412	-3.126	-7.882
Δ v [m/s]	0.1475737	0.368808	3.059222	12.75152	55.05911
s in x-axis [m]	-0.012	-0.177	-12.372	-120.881	-936
s in y-axis [m]	-0.004	-0.14	-17.331	-202.286	>999
s in z-axis [m]	-0.213	-1.208	-18.86	-86.95	-403.975
Δ s [m]	0.21337526	1.2288991	28.445202	251.1815	>999

TABLE 8.2 *Velocity and position, SillyStatus filter used only still on the table.*

Time [s]	5	10	30	60	120
v in x-axis [m/s]	0	0	0	0	0
v in y-axis [m/s]	0	0	0	0	0
v in z-axis [m/s]	0	0	0	0	0
Δ v [m/s]	0	0	0	0	0
s in x-axis [m]	0	0	0	0.001	0.001
s in y-axis [m]	0	0	0	0	0
s in z-axis [m]	0	0	0	-0.001	-0.001
Δ s [m]	0	0	0	0.0014142	0.0014142

TABLE 8.3 *Velocity and position, ANN applied still on the table.*

Time [s]	5	10	30	60	120
Δ v [m/s]	0	0	0	0	0
Δ s [m]	0	0	0	0	0

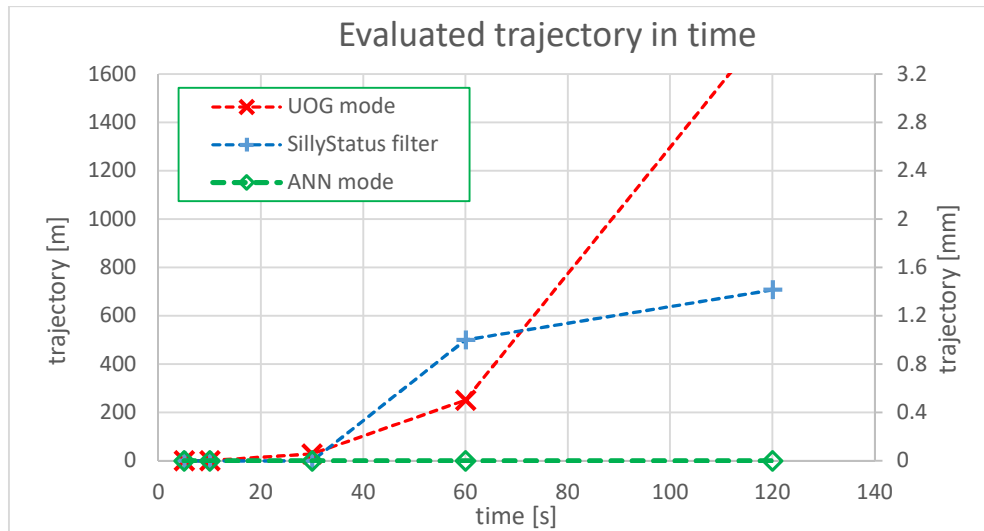


Figure 8.1 Evaluated trajectory in time, the UOG in [m] (left y-axis) and others in [mm] (right y-axis), still on the table.

8.2 Case II: IMU stayed still held in the hand

The results in this case are presented similarly to the results in Chapter 8.1. The SillyStatus filter does not work with satisfactory accuracy. Contrariwise the ANN catches the still states very successfully. After 2 minutes of the measurement, the position is evaluated with the error about 0.17 m in x -axis, 0.13 m in y -axis and 0.08 m in z -axis. The absolute error in positioning with ANN is then about 0.23 m.

The UOG mode results confirm the theory of inertial sensors and show that IMU cannot be used for positioning without auxiliary system.

Figure 8.2 shows the UOG mode curve depicted by the red colour, the SillyStatus mode curve depicted by the blue colour and the ANN aided result curve depicted by the green one. The result of the UOG mode is in meters, the result of the SillyStatus mode and the ANN mode has the second axis on the right hand side in meters. The setting dialog window in Figure 7.3 shows the values of the parameters that were used while the data was processed in SillyStatus mode and which parameters were set while the data was processed with the ANN:

- The “Acc limit” value was set to 0.01
- The “Gyro limit” value was set to 2.
- The “still limit” value was set to 4.
- The “NN status boundary” value was set to -0.99.
- KF for ANN constants: $P = 1$; $Q = 0.5$; $R = 2$.

TABLE 8.4 Velocity and position, UOG mode still in the hand.

Time [s]	5	10	30	60	120
v in x-axis [m/s]	0.046	0.358	3.264	10.72	45.923
v in y-axis [m/s]	0.038	0.197	2.161	9.677	34.569
v in z-axis [m/s]	-0.094	-0.382	-1.48	-3.301	-8.518
Δv [m/s]	0.111337	0.559372	4.184975	14.81416	58.10761
s in x-axis [m]	0.036	0.959	32.02	239.678	>999
s in y-axis [m]	0.027	0.569	22.346	187.27	>999
s in z-axis [m]	-0.08	-1.313	-20.272	-92.479	-443.554
Δs [m]	0.0917878	1.7226175	43.995205	317.91188	>999

TABLE 8.5 Velocity and position, SillyStatus filter used only still in the hand.

Time [s]	5	10	30	60	120
v in x-axis [m/s]	0	-0.006	-0.012	-0.01	0
v in y-axis [m/s]	0	-0.005	-0.005	-0.01	0
v in z-axis [m/s]	0	-0.008	-0.004	-0.006	0
Δv [m/s]	0	0.01118	0.013601	0.015362	0
s in x-axis [m]	0.001	-0.015	-0.203	-0.506	-2.042
s in y-axis [m]	0.001	-0.004	-0.149	-0.421	-1.495
s in z-axis [m]	-0.002	-0.017	-0.162	-0.308	-0.892
Δs [m]	0.002449	0.023022	0.299423	0.726733	2.683366

TABLE 8.6 Velocity and position, ANN applied still in the hand.

Time [s]	5	10	30	60	120
v in x-axis [m/s]	0	0	0	-0.004	0
v in y-axis [m/s]	0	0	0	-0.005	0
v in z-axis [m/s]	0	-0.001	0	-0.001	0
Δv [m/s]	0	0.001	0	0.006481	0
s in x-axis [m]	0	-0.001	-0.004	-0.013	-0.169
s in y-axis [m]	0	0	-0.005	-0.014	-0.128
s in z-axis [m]	-0.001	-0.001	-0.009	-0.017	-0.077
Δs [m]	0.001	0.001414	0.011045	0.025573	0.225553

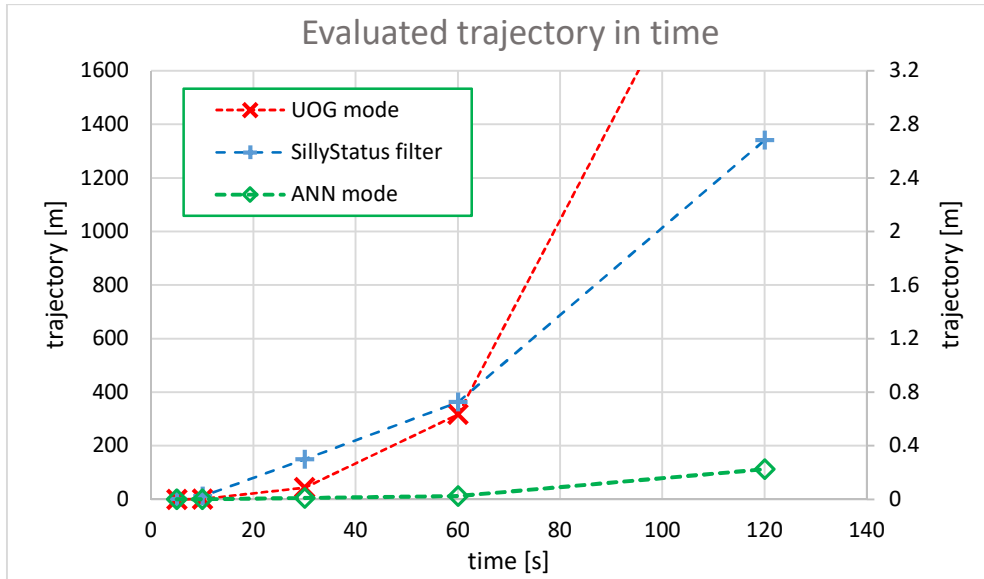


Figure 8.2 Evaluated trajectory in time, the UOG in [m] (left y-axis) and others in [m] (right y-axis) still in the hand.

8.3 Case III: IMU held in the hand during Step and Stop motion

As it was described above, the Step and Stop measurement contain both, steps and still phases. The measurement was performed in Brno, Cerna pole. The true shape of the trajectory is a square with a side length of about $a = 4$ m ($A \Rightarrow B \Rightarrow C \Rightarrow D$). This trajectory repeats for 5 minutes. The detailed trajectory in selected coordinates is shown in Figure 8.3. These data are summarized in TABLE 8.7 Positions of Step and Stop measurement.

The direction to the North (x-axis of the IMU) is positive. The direction to the East (y-axis of the IMU) is negative. That is reason why the minus is present in calculations when the distance in y-axis is computed.

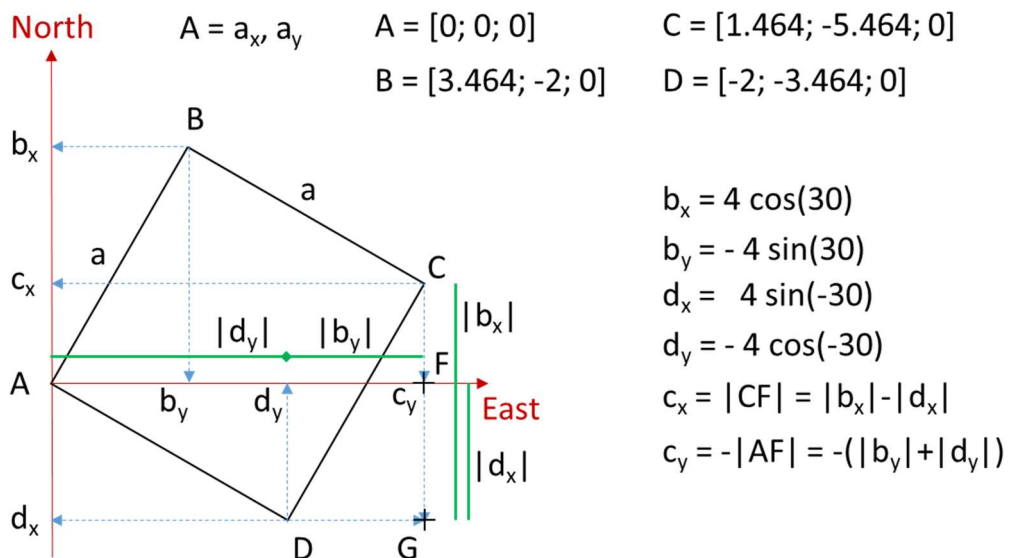


Figure 8.3 Detailed positions of Step and Stop measurement.

TABLE 8.7 Positions of Step and Stop measurement.

square part	length sum [m]	Position vector [m]	azimuth [°]
1. A => B	4	A = (0.000; 0.000; 0.000)	30
2. B => C	8	B = (3.464; 2.000; 0.000)	120
3. C => D	12	C = (1.464; 5.464; 0.000)	210
4. D => A	16	D = (-2.000; 3.464; 0.000)	300

The first three steps are presented in following figures. In this example you can observe the rising inaccuracy in time. The SillyStatus state (red coloured) and neural network output (raw output is green coloured, output after Kalman filtration is blue coloured) is shown in Figure 8.4. The x-axis of all graphs represents time in [s].

In Figure 8.5 the velocity in ENU coordinate system in UOG mode is shown. The corresponding position in time is depicted in Figure 8.8. Again, the velocity in ENU coordinates is shown in Figure 8.6, however the estimation in time is adjusted by SillyStatus filter. The corresponding position in time to this result is depicted in Figure 8.9. The last one mode is shown in Figure 8.7. This is the velocity in ENU coordinates estimated when the ANN is applied. The corresponding position in time is depicted in Figure 8.10. The data was processed with the ANN:

- The “Acc limit” value was set to 0.01.
- The “Gyro limit” value was set to 2.
- The “still limit” value was set to 4.
- The “NN status boundary” value was set to -0.99.
- KF for ANN constants: P = 1; Q = 0.5; R = 2.

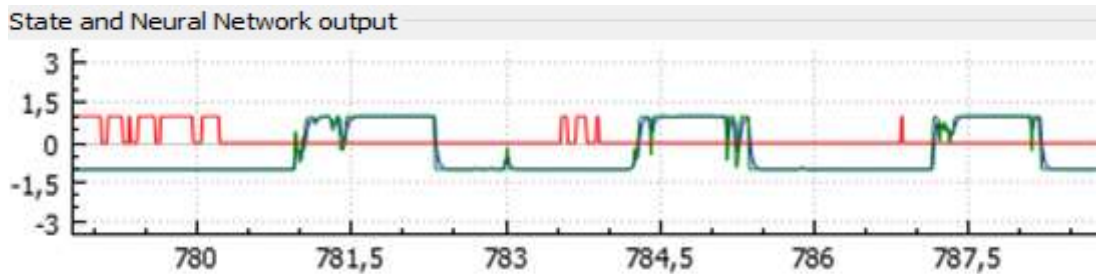


Figure 8.4 SillyStatus state (red) and ANN output state in time [s].

TABLE 8.8 shows the estimated outputs in all three modes, UOG mode, SillyStatus mode and the mode with our ANN in process. The expected values, if could be defined, are shown in the last column. The first part shows the results after three steps. The second part shows the results after 20 steps (approx. 60 seconds of measurement, the position in this time is the same as the initial position, the A point, see Figure 8.3). The third part shows the results after 40 steps (approx. 120 seconds of measurement, the position in this time is the same as the initial position, the A point, see Figure 8.3).

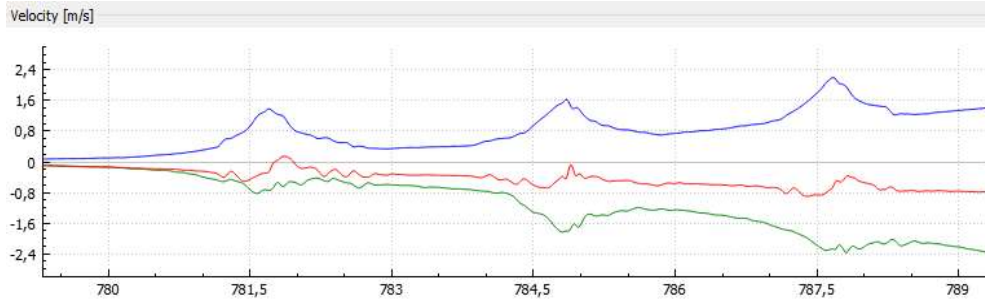


Figure 8.5 ENU velocity estimated in UOG mode, x-axis represents meas. time in [s].

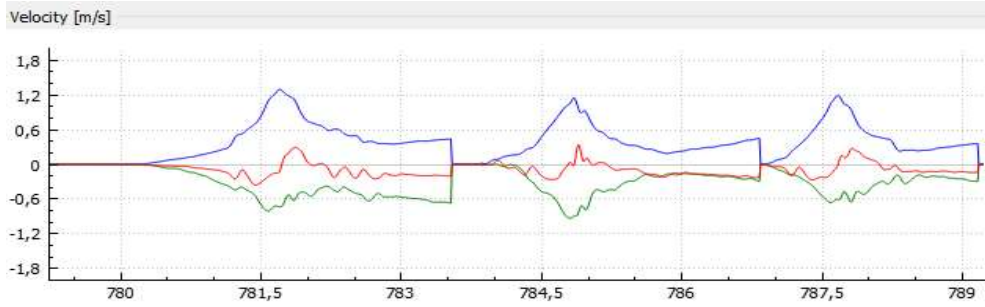


Figure 8.6 ENU velocity estimated with SillyStatus filter in process, x-axis represents meas. time in [s].

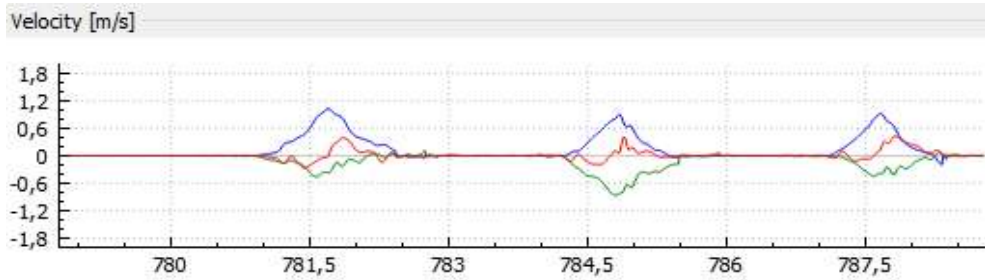


Figure 8.7 ENU velocity estimated with ANN in process, x-axis represents meas. time in [s].

The heading value represents negative value of azimuth of the IMU's x -axis in ENU coordinate system (deviation of north from the x -axis). The table also contains acceleration (\vec{a}), velocity (\vec{v}) and position (\vec{p}) information. In addition, the distance from estimated position to reference (0, 0, 0) in horizontal plane ($|p_{2D}|$) and in 3D ($|p_{3D}|$) is stated. The $|s_{2D}|$ value is the length of the trajectory, the sum of lengths of the particular trajectories. The step length (l_{avg}) is then calculated as the average length of human steps performed during the measuring period. Those values are calculated using equations (8.3.1) - (8.3.4).

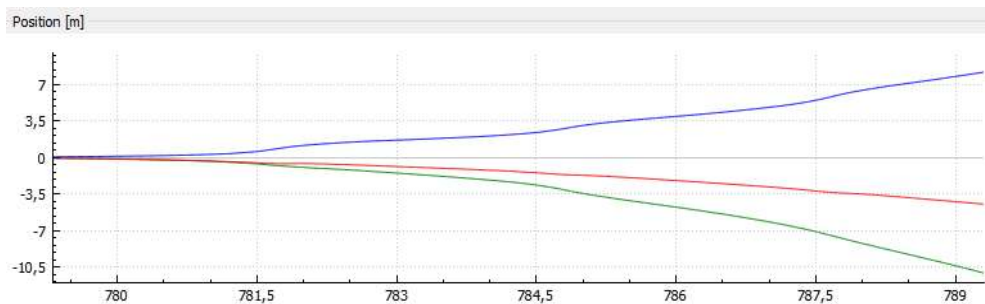


Figure 8.8 ENU position estimated in UOG mode, x-axis represents meas. time in [s].

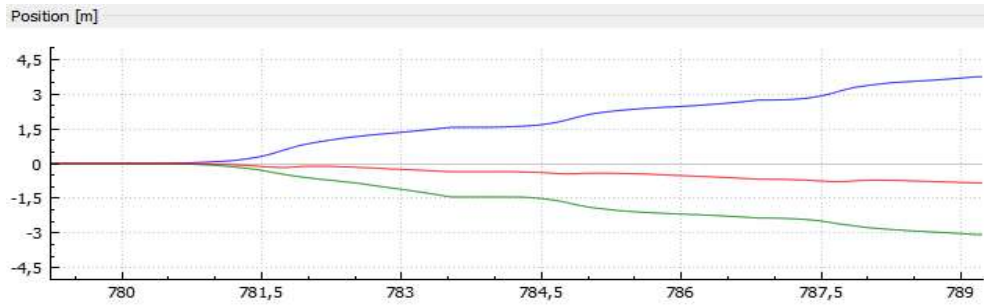


Figure 8.9 ENU position estimated with SillyStatus filter in process, x-axis represents meas. time in [s].

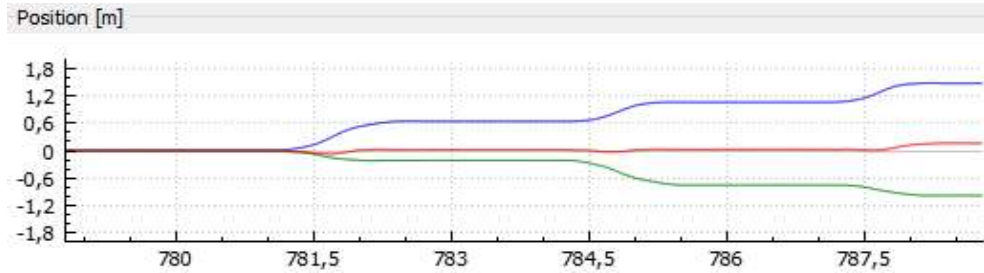


Figure 8.10 ENU position estimated with ANN in process, x-axis represents meas. time in [s].

Inaccuracies arise when rotations by 90 degrees occurs. The applied ANN is not trained to rotations around z-axis without human steps. The graph of distances from the (0, 0, 0) position in ENU coordinate system is shown in Figure 8.11. Figure 8.12 shows the reconstructed 2D trajectory (East-North view) and Figure 8.13 shows the reconstructed 3D trajectory (East-North-Up view) of this measurement using the inertial measurement system only. Graphs show 60 seconds of measurement.

TABLE 8.8 Estimated outputs of the system in particular modes.

mode		UOG	SillyStatus	ANN applied	expected
INITIAL	heading α_0 [°]	-23.35	-26.38	-27.12	-30
	tilt, x-axis φ_0 [°]	-1.71	-2.17	-1.93	NA
	tilt, y-axis θ_0 [°]	-5.92	-6.74	-6.22	NA

mode		UOG	SillyStatus	ANN applied	expected
AFTER THREE STEPS (approx. 13 s)	heading α_1 [°]	34.14	38.85	38.89	30
	tilt, x-axis φ_1 [°]	5.37	6.40	6.23	NA
	tilt, y-axis θ_1 [°]	6.59	7.50	8.34	NA
	a_{1x} [ms ⁻²]	0.54	0.09	-0.05	0.00
	a_{1y} [ms ⁻²]	-0.42	-0.25	-0.18	0.00
	a_{1z} [ms ⁻²]	-0.40	0.26	0.26	0.00
	v_{1x} [ms ⁻¹]	1.42	0.009	-0.001	0.00
	v_{1y} [ms ⁻¹]	-2.37	-0.014	-0.002	0.00
	v_{1z} [ms ⁻¹]	-0.80	0.002	0.003	0.00
	p_{1x} [m]	8.16	3.74	1.95	2.08
	p_{1y} [m]	-11.08	-3.07	-1.05	-1.20
	p_{1z} [m]	-4.48	-0.84	0.01	0.00
	$ p_{2D} $ [m]	8.81	4.84	2.21	2.40
	$ p_{3D} $ [m]	9.89	4.91	2.21	2.40
$ s_{2D} $ [m]	8.81	4.84	2.21	2.40	
step length l_{avg} [m]	2.94	1.61	0.74	0.80	

	mode	UOG	SillyStatus	ANN applied	expected
AFTER 20 STEPS (approx. 60 s)	heading α_0 [°]	18.05	30.53	30.58	30
	tilt, x-axis φ_0 [°]	5.23	2.68	2.89	NA
	tilt, y-axis θ_0 [°]	11.18	8.97	9.21	NA
	a_{2x} [ms ⁻²]	-0.339	-0.022	-0.042	0.00
	a_{2y} [ms ⁻²]	-0.547	-0.193	-0.140	0.00
	a_{2z} [ms ⁻²]	-0.869	-0.64	0.025	0.00
	v_{2x} [ms ⁻¹]	-13.24	14.01	0.006	0.00
	v_{2y} [ms ⁻¹]	-3.67	5.930	-0.006	0.00
	v_{2z} [ms ⁻¹]	-3.62	-5.933	0.002	0.00
	p_{2x} [m]	-307.6	6.95	0.13	0.00
	p_{2y} [m]	-227.1	6.33	-0.015	0.00
	p_{2z} [m]	-105.9	-5.06	0.18	0.00
	$ p_{2D} $ [m]	382.4	9.40	0.13	0.00
	$ p_{3D} $ [m]	396.7	10.68	0.18	0.00
	$ s_{2D} $ [m]	NA	25.71	16.10	16.00
step length l_{avg} [m]	NA	1.29	0.81	0.80	

	mode	UOG	SillyStatus	ANN applied	expected
AFTER 40 STEPS (approx. 113 s)	heading α_0 [°]	37.82	32.24	61.62	60
	tilt, x-axis φ_0 [°]	7.99	8.49	8.64	NA
	tilt, y-axis θ_0 [°]	12.41	9.52	9.73	NA
	a_{3x} [ms ⁻²]	0.029	-0.098	0.00	0.00
	a_{3y} [ms ⁻²]	-0.087	0.400	0.00	0.00
	a_{3z} [ms ⁻²]	-0.176	0.285	0.00	0.00
	v_{3x} [ms ⁻¹]	-25.40	1.99	0.00	0.00
	v_{3y} [ms ⁻¹]	-11.41	0.07	0.00	0.00
	v_{3z} [ms ⁻¹]	-7.44	-0.56	0.00	0.00
	p_{3x} [m]	NA	19.55	0.09	0.00
	p_{3y} [m]	-961	8.84	0.19	0.00
	p_{3z} [m]	-442	-11.45	0.31	0.00
	$ p_{2D} $ [m]	NA	21.46	0.21	0.00
	$ p_{3D} $ [m]	NA	24.32	0.37	0.00
	$ s_{2D} $ [m]	NA	57.05	30.12	32.0
step length l_{avg} [m]	NA	1.43	0.75	0.80	

$$|p_{2D}| = \sqrt{p_{3x}^2 + p_{3y}^2} \quad (8.3.1)$$

$$|p_{3D}| = \sqrt{p_{3x}^2 + p_{3y}^2 + p_{3z}^2} \quad (8.3.2)$$

$$|s_{2D}| = \sqrt{S_x^2 + S_y^2} \quad (8.3.3)$$

$$l_{avg} = \frac{|s_{2D}|}{\text{number of steps}} \quad (8.3.4)$$

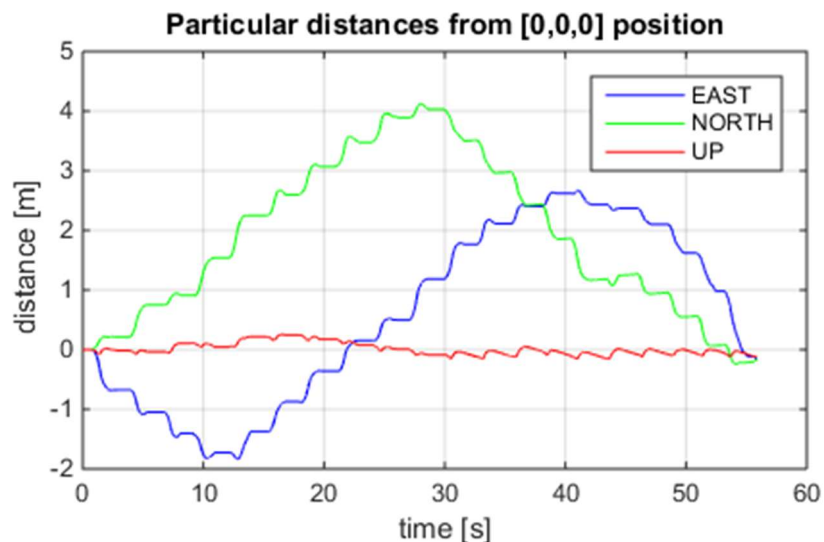


Figure 8.11 Particular distances from (0,0,0) position in ENU coordinates.

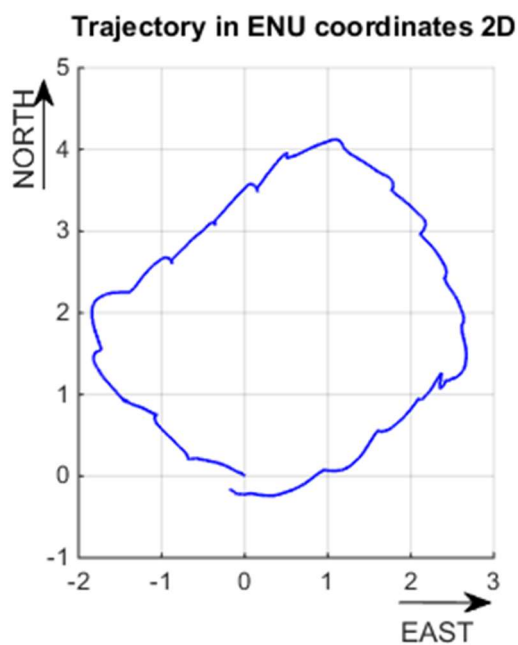


Figure 8.12 Trajectory in ENU coordinates in 2D [m].

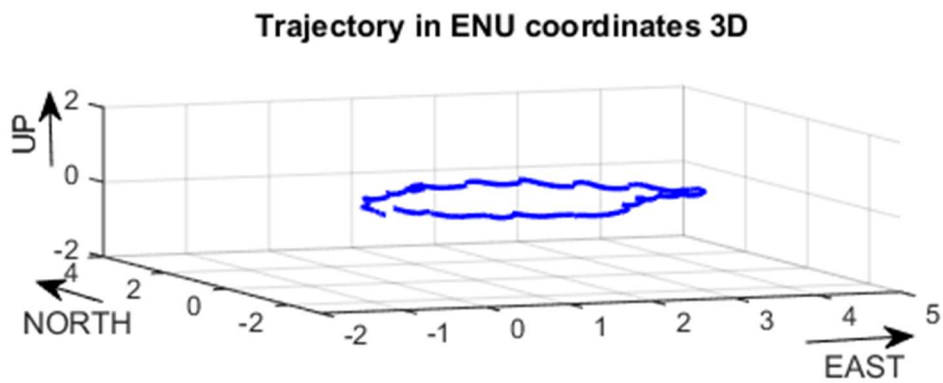


Figure 8.13 Trajectory in ENU coordinates in 3D [m].

8.4 Case IV: IMU held in the hand during discontinuous walking

The source of this measurement is walking interrupted by staying. Following graphs show parts with more frequent status changes only, because the improvement caused by the INS is based on the presence of still phases.

Next figures capture the velocity (Figure 8.14) and position (Figure 8.15) of the two steps in the North direction and short still phase repeated three times. The blue curve represents the x -axis, the green one y -axis the red one z -axis. After the walking part shown in graphs, proposed ANN caught the still phase clearly, nevertheless it does not improve the positioning during the walking. The position after six steps was 5.28 m in the North direction; 0.3 m in the West direction and 0.8 m in the Down direction.

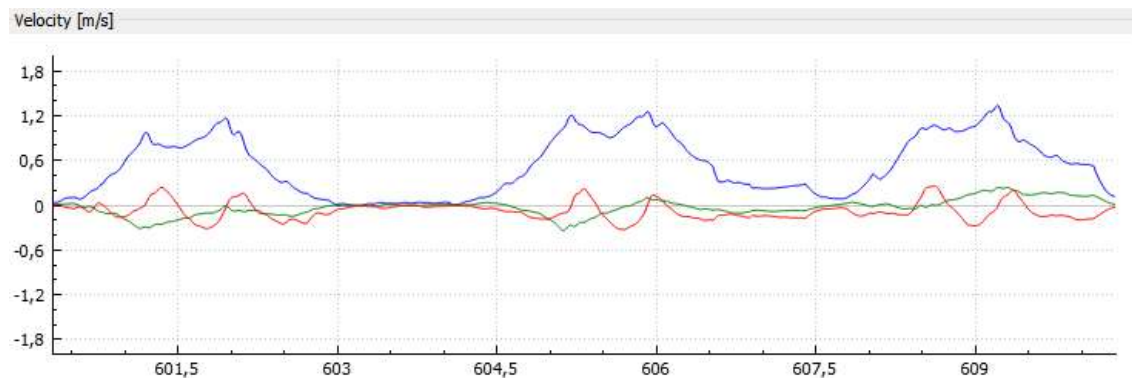


Figure 8.14 ENU velocity estimated with ANN in process [ms^{-1}] in time [s] - interrupted walking.

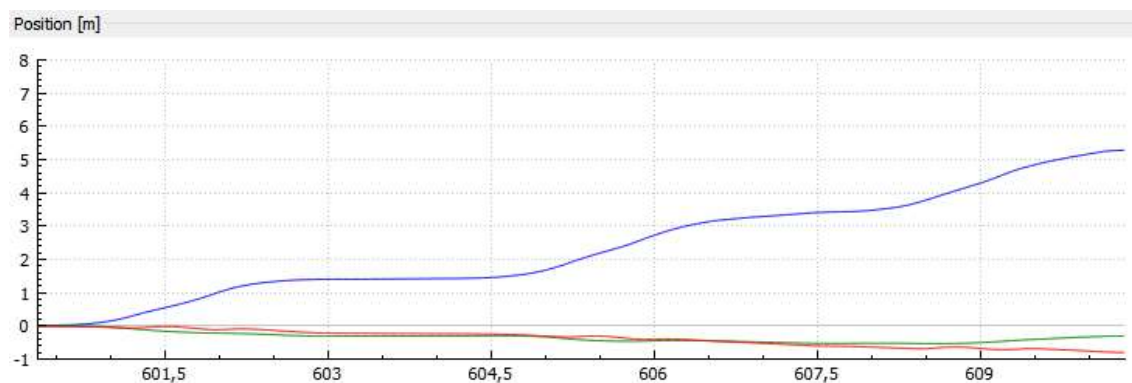


Figure 8.15 ENU position estimated with ANN in process [m] in time[s] - interrupted walking.

Whole measurement trajectory is shown in Figure 8.16. The movement in the North direction consists of double steps (repeated five times) and the movement in the South direction, after the rotation by 180 degrees, consists of continuous three steps and still phase, repeated three times. The start position was defined as (0;0;0) m and expected final position was (1;0;0) m.

The resulted final position was (2.6;-1.55;-2.0) m. The length of the measurement was 50 seconds. The error rises significantly with the length of the continuous walking.

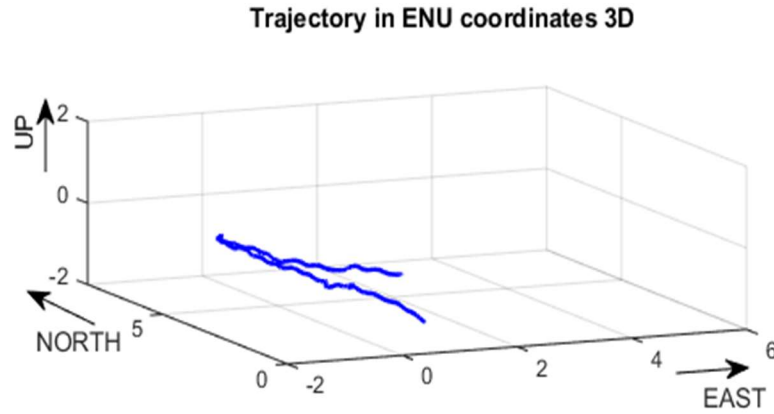


Figure 8.16 Trajectory in ENU coordinates, 3D in [m] - interrupted walking.

8.5 Case V: IMU held in the hand during fluent walking

As it is clear from the principle of the ANN implementation, the improvement is not achieved when the walking is present only. This case explains it illustratively. Further figures show linear acceleration in ENU coordinate frame (Figure 8.17), computed position (Figure 8.18), and decision on the state of the IMU (Figure 8.19). At the beginning (up to 191.2 s) the still phase is present and successfully determined by the ANN. After that the motion continues with fluent walking - the ANN indicate it successfully and of course further positioning process is not improved by the ANN.

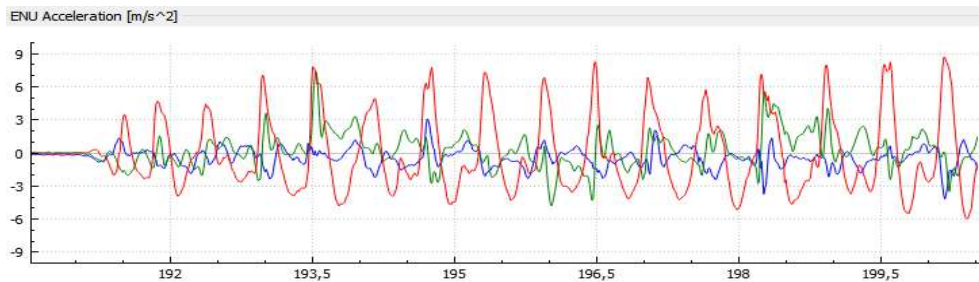


Figure 8.17 Linear acceleration in ENU coordinates in [ms⁻²] in time [s], continuous walking.

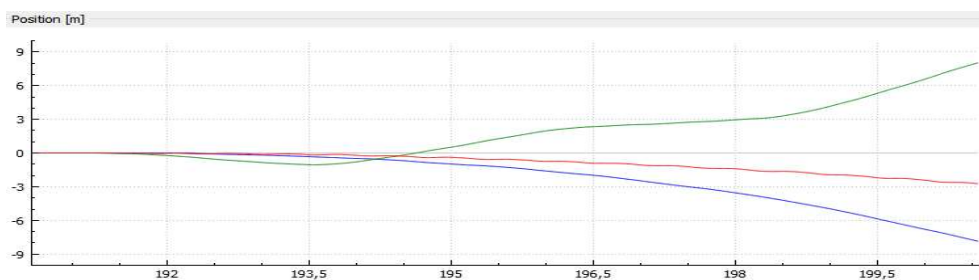


Figure 8.18 Position in ENU coordinates in [m] in time [s], continuous walking.

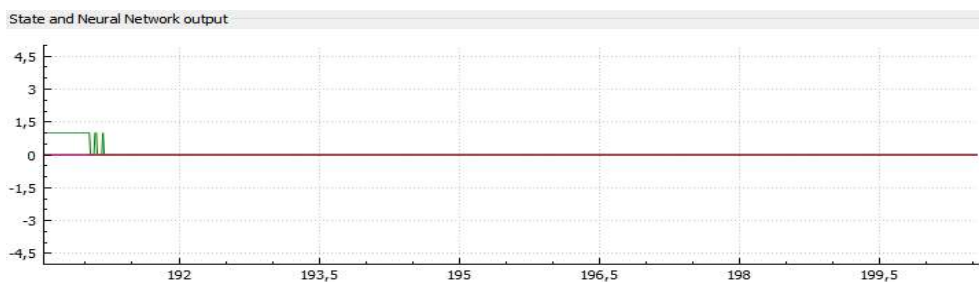


Figure 8.19 Combined SillyStatus state and ANN decision on the IMU state (green) in time.

9 CONCLUSIONS

In proposed dissertation thesis, the processing of inertial navigation sensor data is presented. As the new approach method I decided to estimate the state of the IMU by an artificial neural network without any support of auxiliary or global positioning system. It ensures that the data from the inertial sensors are processed typically (with the integration disadvantages) only for a vital period. The orientation of the IMU is fully derivate from inertial sensors.

The correction of the IMU orientation is performed during data processing when the ANN decides that the IMU is still. That leads to more accurate positioning based on DR, regardless of the environment (indoor, outdoor, underground, etc.). As the IMU hardware, Arduino UNO was chosen in combination with ST Nucleo expansion board, which contains all used inertial sensors.

Special positioning system software called Tracker was developed in C++ programming language using Qt framework. It also offers graphical environment for the user. It process the data from the IMU and presents various intermediate and final results. The system also allows to record data into a file in adjustable format – raw/calibrated/derotated sensor data, Euler angles, heading, velocity and position in all available modes. A window with 3D IMU model is also available.

Proposed ANN was designed in MATLABTM software and estimates the state of the IMU based on the previous 40 values from inertial sensors, the type of the ANN is time-delayed feed-forward. It does not take the data from magnetometer into account, because of the magnetic field typically extremely fluctuates. The output of the ANN defines the state of the IMU – „walking“ or „staying still“, which is applied in data processing to improve positioning.

Such a system works very precisely in case that the IMU stays still on the table or stays still in the hand. In those cases, the error in positioning reached about 2 millimetres in the case the IMU was lying on the table and about 20 centimetres in case the IMU was held in the hand, after 2 minutes of acquisition.

The very interesting results were achieved when the IMU was held in the hand and the user performed a walk that often changes with still phases. Such a motion can be seen for example in a museum or in an art gallery. In these cases proposed system achieves very small positioning errors compared to the systems based purely on DR method. As shown in Chapter 8.3 the INS achieved the error of only 2 meters after 2 minutes of measurement in 2D (horizontal positioning). The error in vertical z-axis reached up to 5 meters and that was caused by subtraction of the inaccurately determined earth's gravitational acceleration constant.

In situations when the ANN decides that the IMU is still, the system is recalibrated and the cumulative error caused by integration is reset. Thus the position during discontinuous walking is effectively estimated with low error. When the walking motion is present during the measurement only, this method fails and the INS works as a simple DR system (however, in a real world a man must stop anytime).

In this dissertation thesis, proposed method based on ANN state recognition has been successfully validated by experiments focused on pedestrian movements. Anyway, more applications can be found in a human life in which this method could improve positioning, for example in specific professions, military applications or different types of vehicles. It opens new opportunities in future research for specific applications where the suitable artificial neural network structure have to be investigated and properly trained or modified with wider classification group (more types of movements).

APPENDIX A COORDINATE TRANSFORMATIONS

A.1 Representation of transformed vectors

a vector \mathbf{v} in XYZ coordinates:

$$\mathbf{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}, \quad (\text{A.1})$$

the same vector \mathbf{v} in UVW coordinates:

$$\mathbf{v} = \begin{bmatrix} v_u \\ v_v \\ v_w \end{bmatrix}, \quad (\text{A.2})$$

then, in any Cartesian system apply:

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = C_{XYZ}^{UVW} \begin{bmatrix} v_u \\ v_v \\ v_w \end{bmatrix}. \quad (\text{A.3})$$

C_{NED}^{RPY} then denotes the coordinate transformation matrix from vehicle body-fixed roll-pitch-yaw (RPY) frame coordinates to earth-fixed north-east-down (NED) coordinates.

A.2 Unit coordinate vectors

The components of a vector in either coordinate system can be expressed in terms of the vector components along unit vectors parallel to the respective coordinate axes:

$$\begin{aligned} \mathbf{v} &= v_x \cdot \vec{\mathbf{1}}_x + v_y \cdot \vec{\mathbf{1}}_y + v_z \cdot \vec{\mathbf{1}}_z \\ &= v_u \cdot \vec{\mathbf{1}}_u + v_v \cdot \vec{\mathbf{1}}_v + v_w \cdot \vec{\mathbf{1}}_w. \end{aligned} \quad (\text{A.4})$$

A.3 Direction cosines

The respective components can be also represented as a dot products, in matrix form:

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{1}}_x^T \cdot \vec{\mathbf{1}}_u & \vec{\mathbf{1}}_x^T \cdot \vec{\mathbf{1}}_v & \vec{\mathbf{1}}_x^T \cdot \vec{\mathbf{1}}_w \\ \vec{\mathbf{1}}_y^T \cdot \vec{\mathbf{1}}_u & \vec{\mathbf{1}}_y^T \cdot \vec{\mathbf{1}}_v & \vec{\mathbf{1}}_y^T \cdot \vec{\mathbf{1}}_w \\ \vec{\mathbf{1}}_z^T \cdot \vec{\mathbf{1}}_u & \vec{\mathbf{1}}_z^T \cdot \vec{\mathbf{1}}_v & \vec{\mathbf{1}}_z^T \cdot \vec{\mathbf{1}}_w \end{bmatrix} \cdot \begin{bmatrix} v_u \\ v_v \\ v_w \end{bmatrix} \stackrel{\text{def}}{=} C_{XYZ}^{UVW} \cdot \begin{bmatrix} v_u \\ v_v \\ v_w \end{bmatrix}. \quad (\text{A.5})$$

dot product of unit vector satisfy the cosine rule: $v^T w = |v| \cdot |w| \cdot \cos(\theta_{ab})$, where θ_{ab} is the angle between vectors v and w . The coordinate transformation matrix can be then written as:

$$C_{XYZ}^{UVW} = \begin{bmatrix} \cos(\theta_{xu}) & \cos(\theta_{xv}) & \cos(\theta_{xw}) \\ \cos(\theta_{yu}) & \cos(\theta_{yv}) & \cos(\theta_{yw}) \\ \cos(\theta_{zu}) & \cos(\theta_{zv}) & \cos(\theta_{zw}) \end{bmatrix}. \quad (\text{A.6})$$

The angles determination do not depend on the order of the direction vectors ($\theta_{ab} = \theta_{ba}$), the inverse transformation matrix is simply transposition of the forward coordinate transformation matrix:

$$C_{UVW}^{XYZ} = \begin{bmatrix} \cos(\theta_{xu}) & \cos(\theta_{xv}) & \cos(\theta_{xw}) \\ \cos(\theta_{yu}) & \cos(\theta_{yv}) & \cos(\theta_{yw}) \\ \cos(\theta_{zu}) & \cos(\theta_{zv}) & \cos(\theta_{zw}) \end{bmatrix}^T = (C_{XYZ}^{UVW})^T. \quad (\text{A.7})$$

A.4 RPY/ENU and RPY/NED transformations

The resulting unit vectors of the roll, pitch and yaw axes in ENU coordinates are defined as:

$$\vec{\mathbf{i}}_R = \begin{bmatrix} \sin(Y) \cos(P) \\ \cos(Y) \cos(P) \\ \sin(P) \end{bmatrix}, \quad (\text{A.8})$$

$$\vec{\mathbf{i}}_P = \begin{bmatrix} \cos(R) \cos(Y) + \sin(R) \sin(Y) \sin(P) \\ -\cos(R) \sin(Y) + \sin(R) \cos(Y) \sin(P) \\ -\sin(R) \cos(P) \end{bmatrix}, \quad (\text{A.9})$$

$$\vec{\mathbf{i}}_Y = \begin{bmatrix} -\sin(R) \cos(Y) + \cos(R) \sin(Y) \sin(P) \\ \sin(R) \sin(Y) + \cos(R) \cos(Y) \sin(P) \\ -\cos(R) \cos(P) \end{bmatrix}. \quad (\text{A.10})$$

The unit vectors of the east, north and up axes in RPY coordinates are defined as:

$$\vec{\mathbf{i}}_E = \begin{bmatrix} \sin(Y) \cos(P) \\ \cos(R) \cos(Y) + \sin(R) \sin(Y) \sin(P) \\ -\sin(R) \cos(Y) + \cos(R) \sin(Y) \sin(P) \end{bmatrix}, \quad (\text{A.11})$$

$$\vec{\mathbf{I}}_N = \begin{bmatrix} \cos(Y)\cos(P) \\ -\cos(R)\sin(Y) + \sin(R)\cos(Y)\sin(P) \\ \sin(R)\sin(Y) + \cos(R)\cos(Y)\sin(P) \end{bmatrix}, \quad (\text{A.12})$$

$$\vec{\mathbf{I}}_U = \begin{bmatrix} \sin(P) \\ -\sin(R)\cos(P) \\ -\cos(R)\cos(P) \end{bmatrix}. \quad (\text{A.13})$$

The rotation from RPY coordinates to NED coordinates is determined by:

$$C_{NED}^{RPY} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) \end{bmatrix}, \quad (\text{A.14})$$

$$C_{NED}^{RPY} = \begin{bmatrix} \cos(\psi) \cdot \cos(\theta) & -\sin(\psi) \cdot \cos(\varphi) + \cos(\psi) \cdot \sin(\theta) \cdot \sin(\varphi) & \sin(\psi) \cdot \sin(\varphi) + \cos(\psi) \cdot \sin(\theta) \cdot \cos(\varphi) \\ \sin(\psi) \cdot \cos(\theta) & \cos(\psi) \cdot \cos(\varphi) + \sin(\psi) \cdot \sin(\theta) \cdot \sin(\varphi) & -\cos(\psi) \cdot \sin(\varphi) + \sin(\psi) \cdot \sin(\theta) \cdot \cos(\varphi) \\ -\sin(\theta) & \cos(\theta) \cdot \sin(\varphi) & \cos(\theta) \cdot \cos(\varphi) \end{bmatrix} \quad (\text{A.15})$$

The rotation from RPY coordinates to ENU coordinates is determined by:

$$C_{ENU}^{RPY} = \begin{bmatrix} \sin(\psi) \cdot \cos(\theta) & \cos(\psi) \cdot \cos(\varphi) + \sin(\psi) \cdot \sin(\theta) \cdot \sin(\varphi) & -\cos(\psi) \cdot \sin(\varphi) + \sin(\psi) \cdot \sin(\theta) \cdot \cos(\varphi) \\ \cos(\psi) \cdot \cos(\theta) & -\sin(\psi) \cdot \cos(\varphi) + \cos(\psi) \cdot \sin(\theta) \cdot \sin(\varphi) & \sin(\psi) \cdot \sin(\varphi) + \cos(\psi) \cdot \sin(\theta) \cdot \cos(\varphi) \\ \sin(\theta) & -\cos(\theta) \cdot \sin(\varphi) & -\cos(\theta) \cdot \cos(\varphi) \end{bmatrix} \quad (\text{A.16})$$

A.5 ENU/ECEF and NED/ECEF transformations

The unit vectors in local *north*, *east* and *down* directions, as expressed in ECEF Cartesian coordinates are defined as:

$$\vec{\mathbf{I}}_N = \begin{bmatrix} -\cos(\theta) \sin(\phi_{geo \ det \ ic}) \\ -\sin(\theta) \sin(\phi_{geo \ det \ ic}) \\ \cos(\phi_{geo \ det \ ic}) \end{bmatrix}, \quad (\text{A.17})$$

$$\vec{\mathbf{I}}_E = \begin{bmatrix} \sin(\theta) \\ \cos(\theta) \\ 0 \end{bmatrix}, \quad (\text{A.18})$$

$$\vec{\mathbf{I}}_D = \begin{bmatrix} -\cos(\theta) \cos(\phi_{geo \ det \ ic}) \\ -\sin(\theta) \cos(\phi_{geo \ det \ ic}) \\ -\sin(\phi_{geo \ det \ ic}) \end{bmatrix}. \quad (\text{A.19})$$

And the unit vectors in the ECEF X, Y and Z directions, as expressed in NED coordinates are defined as:

$$\vec{\mathbf{i}}_X = \begin{bmatrix} -\cos(\theta) \sin(\phi_{geo\ det\ ic}) \\ -\sin(\theta) \\ -\cos(\theta) \cos(\phi_{geo\ det\ ic}) \end{bmatrix}, \quad (\text{A.20})$$

$$\vec{\mathbf{i}}_Y = \begin{bmatrix} -\sin(\theta) \sin(\phi_{geo\ det\ ic}) \\ \cos(\theta) \\ -\sin(\theta) \cos(\phi_{geo\ det\ ic}) \end{bmatrix}, \quad (\text{A.21})$$

$$\vec{\mathbf{i}}_Z = \begin{bmatrix} \cos(\phi_{geo\ det\ ic}) \\ 0 \\ -\sin(\phi_{geo\ det\ ic}) \end{bmatrix}. \quad (\text{A.22})$$

The rotation from NED coordinates to ECEF coordinates is determined by:

$$C_{ECEF}^{NED} = \begin{bmatrix} -\cos(\theta) \sin(\phi_{geodetic}) & -\sin(\theta) & -\cos(\theta) \cos(\phi_{geodetic}) \\ -\sin(\theta) \sin(\phi_{geodetic}) & \cos(\theta) & -\sin(\theta) \cos(\phi_{geodetic}) \\ \cos(\phi_{geodetic}) & 0 & -\sin(\phi_{geodetic}) \end{bmatrix}. \quad (\text{A.23})$$

The rotation from ENU coordinates to ECEF coordinates is determined by:

$$C_{ECEF}^{ENU} = \begin{bmatrix} -\sin(\theta) & -\cos(\theta) \sin(\phi_{geodetic}) & \cos(\theta) \cos(\phi_{geodetic}) \\ \cos(\theta) & -\sin(\theta) \sin(\phi_{geodetic}) & \sin(\theta) \cos(\phi_{geodetic}) \\ 0 & \cos(\phi_{geodetic}) & \sin(\phi_{geodetic}) \end{bmatrix}. \quad (\text{A.24})$$

The rotation from ECEF coordinates to NED coordinates is determined by:

$$C_{NED}^{ECEF} = \begin{bmatrix} -\cos(\theta) \sin(\phi_{geodetic}) & -\sin(\theta) \sin(\phi_{geodetic}) & \cos(\phi_{geodetic}) \\ -\sin(\theta) & \cos(\theta) & 0 \\ -\cos(\theta) \cos(\phi_{geodetic}) & -\sin(\theta) \cos(\phi_{geodetic}) & -\sin(\phi_{geodetic}) \end{bmatrix}. \quad (\text{A.25})$$

The rotation from ECEF coordinates to ENU coordinates is determined by:

$$C_{ENU}^{ECEF} = \begin{bmatrix} -\sin(\theta) & \cos(\theta) & 0 \\ -\cos(\theta) \sin(\phi_{geodetic}) & -\sin(\theta) \sin(\phi_{geodetic}) & \cos(\phi_{geodetic}) \\ \cos(\theta) \cos(\phi_{geodetic}) & \sin(\theta) \cos(\phi_{geodetic}) & \sin(\phi_{geodetic}) \end{bmatrix}. \quad (\text{A.26})$$

A.6 Composition of coordinate transformations

When A, B and C represent different coordinate frames, next composition rule applies:

$$C_C^B \cdot C_B^A = C_C^A. \quad (\text{A.27})$$

APPENDIX B MEASUREMENT CONDITIONS

B.1 Geographic coordinates

Stamicova, Brno - Kohoutovice, Czech Republic:

Latitude: 49°11'42" N

Longitude: 16°36'28" E

Elevation above sea level: 361m

Coordinates of the place in decimal degrees:

Latitude: 49.1952200 N

Longitude: 16.6079600 E

Tomanova, Brno - Cerna pole, Czech Republic:

Latitude: 49°12'24" N

Longitude: 16°37'04 " E

Elevation above sea level: 239 m

Coordinates of the place in decimal degrees:

Latitude: 49.2078766 N

Longitude: 16.6193204 E

Volejnikova, Brno - Cerna pole, Czech Republic:

Latitude: 49°12'31" N

Longitude: 16°37'19" E

Elevation above sea level: 233 m

Coordinates of the place in decimal degrees:

Latitude: 49.2085308 N

Longitude: 16.6219269 E

Technicka, Brno - Kralovo Pole, Czech Republic:

Latitude: 49°13'37" N

Longitude: 16°34'29" E

Elevation above sea level: 287 m

Coordinates of the place in decimal degrees:

Latitude: 49.2271495 N

Longitude: 16.5726303 E

B.2 Temperature stability

Accelerometer and gyroscope sensor included in chip LSM6DS0 are sensitive to temperature changes. Due to this fact, the additional temperature calibration have to be performed assuming that the temperature changes occurs during the measurement.

The gyroscope reaction is captured in Figure B.1. The temperature dropped by 5 degrees after about 550 seconds of measurement. The y-axis of the graph represents rotation rate in [deg/s], x-axis represents the time [s]. The IMU stood still on the table during this measurement.

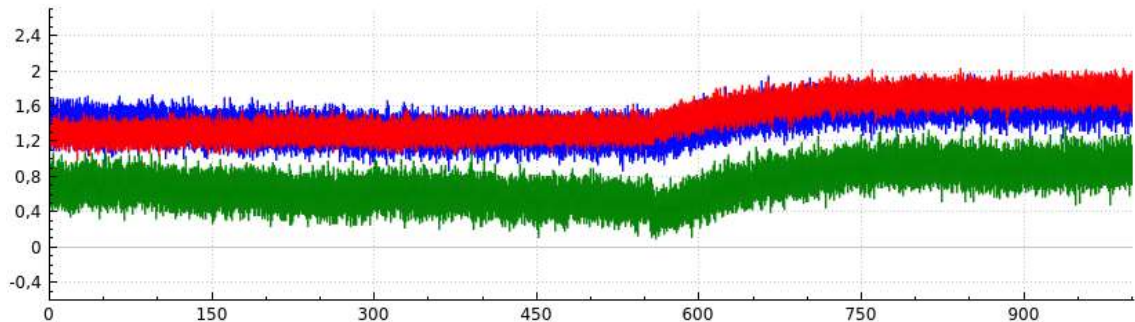


Figure B.1 Change in the measured rotation rate [deg/s] when the temperature drops down (in 550 s) in time [s].

B.3 Calibration parameters

```
//Tomanova, exapmle
magMin = QVector3D(-0.650, -0.640, 0.220);
magMax = QVector3D( 0.325,  0.320, 1.150);

//Local magnetic field
MagLocal = 415.230; //Stamicova [mGauss];
MagLocal = 489.151; //Tomanova [mGauss];
MagLocal = 439.696; //Volejnikova [mGauss];
MagLocal = 429.190; //Technicka [mGauss];

//Local declination
MagDeclination = 4.21; //Stamicova [deg];
MagDeclination = 4.21; //Tomanova [mGauss];
MagDeclination = 4.21; //Volejnikova [mGauss];
MagDeclination = 4.20; //Technicka [mGauss];
```

APPENDIX C HARDWARE SETTING

C.1 Hardware description

MinIMU-9 v2 Gyro, Accelerometer, and Compass

L3GD20 – MEMS chip containing and 3-axis gyroscope

The range of maximum and minimum value of measured signals is user-defined:

- Gyroscope available scales: $\pm 250, \pm 500, \pm 2000$ deg/s

LSM303DLHC – MEMS chip containing 3-axis magnetometer and accelerometer

The range of maximum and minimum value of measured signals is user-defined:

- Accelerometer available scales: $\pm 2, \pm 4, \pm 8, \pm 16$ g
- Magnetometer available scales: $\pm 1.3, \pm 1.9, \pm 2.5, \pm 4.0, \pm 4.7, \pm 5.6, \pm 8.1$ gauss

Sensors work within the temperature range of -40 °C to 85 °C.

The IMU logs measured data to its own memory and sends data to PC via serial link after the measurement. The transmission is performed after “send” button is pressed. The sample rate of logging data reaches 22 sps (using this construction).

Motion MEMS and environmental sensor expansion board X-NUCLEO-IKS01A1

LSM6DS0 – MEMS chip containing and 3-axis gyroscope and accelerometer

The range of maximum and minimum value of measured signals is user-defined:

- Gyroscope available scales: $\pm 245, \pm 500, \pm 2000$ deg/s
- Accelerometer available scales: $\pm 2, \pm 4, \pm 8$ g

LIS3MDL – MEMS chip containing 3-axis magnetometer

- Magnetometer available scales: $\pm 4, \pm 8, \pm 12, \pm 16$ gauss

LPS25HB* – barometer MEMS pressure sensor

- Barometer absolute digital output: $260 - 1260$ hPa

HTS221 – capacitive digital relative humidity and temperature sensor

The expansion board is compatible with Arduino Uno. Measured data are sent by BT or by USB cable to a PC. Processing is available real time or there is ability to save measured data for further processing based on the user defined settings in the software Tracker. The IMU transmits data to PC via Bluetooth or USB cable during the measurement. The sample rate of incoming data can reach 160 sps.

C.2 Sensor settings

X-NUCLEO-IKS01A1 board sensors are set:

Angular rate sensor control register

CTRL_REG1_G B00010000	ODR_G [2:0]	100	ODR = 238 Hz; cutoff (LPF1) = 76 Hz
	FS_G [1:0]	01	gyro scale ± 500 deg/s
	BW_G [1:0]	10	cutoff (LPF2) = 63 Hz – not used
CTRL_REG2_G B00010001	INT_SEL [1:0]	00	default
	OUT_SEL [1:0]	00	default
CTRL_REG3_G B 00010010	LP_mode	0	low-power disable
	HP_EN	0	HPF disabled
	HPCF_G [3:0]	0000	high-pass filter cutoff = 15 Hz

Linear acceleration sensor control register

CTRL_REG5_XL B00011111	DEC [0:1]	00	no decimation on OUT REG and FIFO
	Zen_XL Yen_XL Xen_XL		x, y, z axis enable
CTRL_REG6_XL B00100000	ODR_XL [2:0]	000	ODR = 238 Hz (same as gyro ODR)
	FS_XL [1:0]	00	accelerometer full-scale ± 2 g
	BW_SCAL ODR	0	bandwidth determined by ODR selection
	BW_XL [1:0]	00	anti-aliasing filter bandwidth = 408 Hz

APPENDIX C

CTRL_REG7_XL B00100001	HR	1	High resolution mode enabled
	DCF[1:0]	01	LP cutoff freq. = ODR/100 Hz
	FDS	0	internal filter bypassed
	HPIS1	0	filter bypassed

Magnetic field sensor control register

CTRL_REG1 B00010000	TEMP_EN	0	temperature sensor disabled
	OM[1:0]	10	high-performance mode
	DO[2:0]	111	ODR = 80 Hz
	FAST_ODR	0	fast ODR disabled
	ST	0	self-test disabled
CTRL_REG2 B00010001	FS[1:0]	00	magnetometer full-scale ± 4 gauss
	REBOOT	0	normal mode reboot memory
	SOFT_RST	0	default
CTRL_REG3 B00010010	LP	0	default
	SIM	0	4-wire interface
	MD[1:0]	01	Single-conversion mode
CTRL_REG4 B00010011	OMZ[1:0]	10	high-performance mode
	BLE	0	data LSB at lower address
CTRL_REG5 B00010100	FAST_READ	0	FAST_READ disabled
	BDU	1	continuous update

C.3 I²C Communication

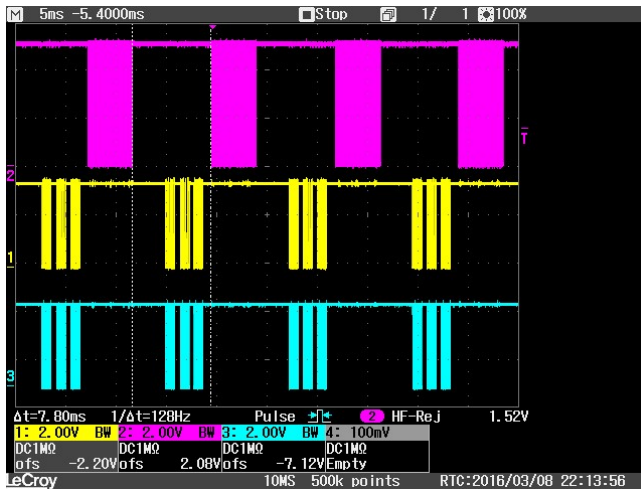


Figure C.2 Four Arduino communication cycles, consequently. TX, SDA, SCL.

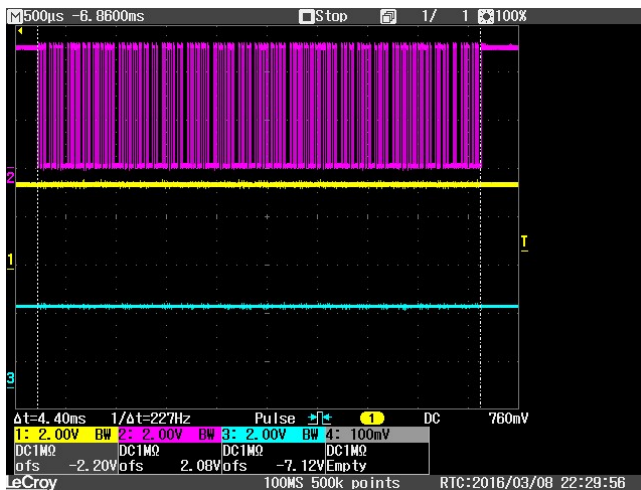


Figure C.3 Detail of RS232 Communication cycle, consequently. TX, SDA, SCL.

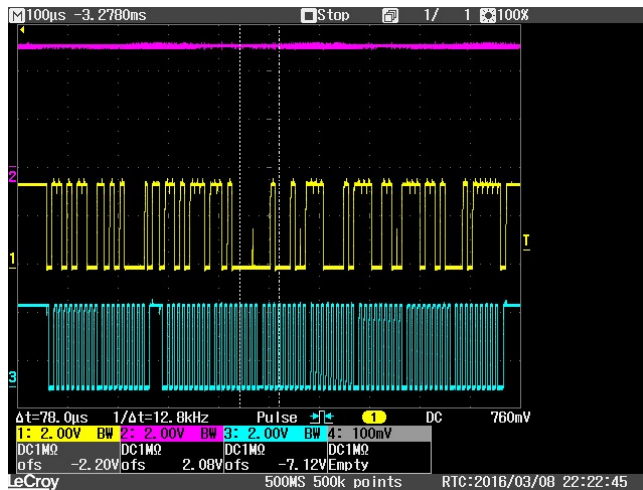


Figure C.4 Detail of I²C communication cycle, consequently. TX, SDA, SCL.

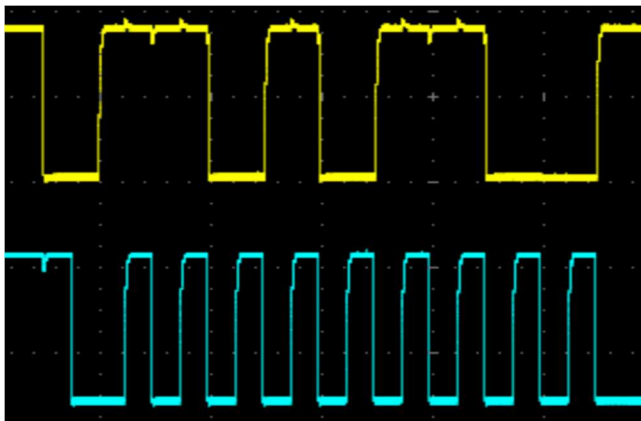


Figure C.5 Transmission of one byte via I²C. SDA, SCL

APPENDIX D FIRMWARE

```
#include <Wire.h>

void readFrom(int address, byte sub, int count)
{
    if (count > 1) sub += B10000000;

    Wire.beginTransmission(address);
    Wire.write(sub);
    Wire.endTransmission();

    Wire.requestFrom(address, count);
}

void writeTo(int address, byte sub, byte data)
{
    Wire.beginTransmission(address);
    Wire.write(sub);
    Wire.write(data);
    Wire.endTransmission();
}
```

APPENDIX D

```
void setupMag()
{
  writeTo(30, B00100000, B01011100); // CTRL_REG1 20h
  writeTo(30, B00100001, B00000000); // CTRL_REG2 21h
  writeTo(30, B00100010, B00000001); // CTRL_REG3 22h
  writeTo(30, B00100011, B00001000); // CTRL_REG4 23h
  writeTo(30, B00100100, B01000000); // CTRL_REG5 24h
}

void setupAG()
{
  writeTo(107, B00010000, B10001010); // CTRL_REG1_G 10h
  writeTo(107, B00010001, B00000000); // CTRL_REG2_G 11h
  writeTo(107, B00010010, B00000000); // CTRL_REG3_G 12h
  writeTo(107, B00011111, B00111000); // CTRL_REG5_XL 1Fh
  writeTo(107, B00100000, B00000000); // CTRL_REG6_XL 20h
  writeTo(107, B00100001, B10100000); // CTRL_REG7_XL 21h
}

void setup()
{
  Wire.begin();
  Serial.begin(115200);

  setupMag();
  setupAG();
}

void loop()
{
  readFrom(30, B00101000, 6);
  while (Wire.available())
  {
    short c = Wire.read();
    c += (Wire.read() << 8);
    Serial.print(c, DEC);
    Serial.print(" ");
  }

  readFrom(107, B00011000, 6);
  while (Wire.available())
  {
    short c = Wire.read();
    c += (Wire.read() << 8);
    Serial.print(c, DEC);
    Serial.print(" ");
  }

  readFrom(107, B00101000, 6);
  while (Wire.available())
  {
    short c = Wire.read();
    c += (Wire.read() << 8);
    Serial.print(c, DEC);
    Serial.print(" ");
  }

  Serial.print("\n");
  delay(7);
}
```

APPENDIX E DATA PROCESSING

Further figures show all successive phases of processing that are graphically available in our app Tracker during the processing of the incoming sensor signal.

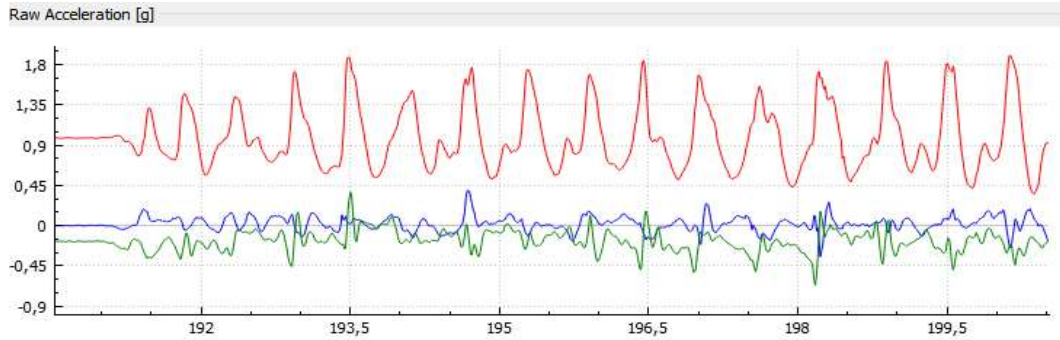


Figure E.6 Raw acceleration [g] in time [s].

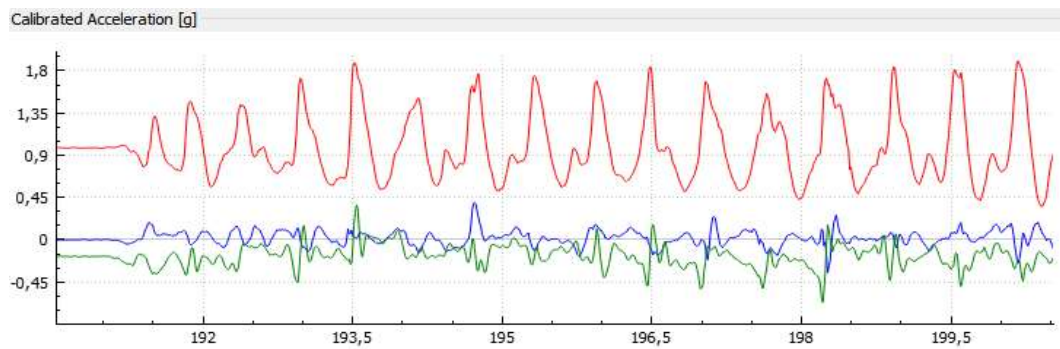


Figure E.7 Calibrated acceleration [g] in time [s].

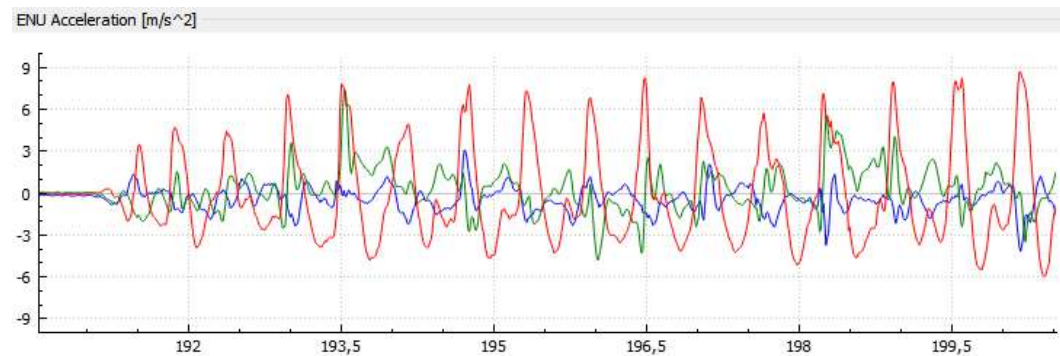


Figure E.8 ENU (linear) acceleration [m·s⁻²] in time [s].

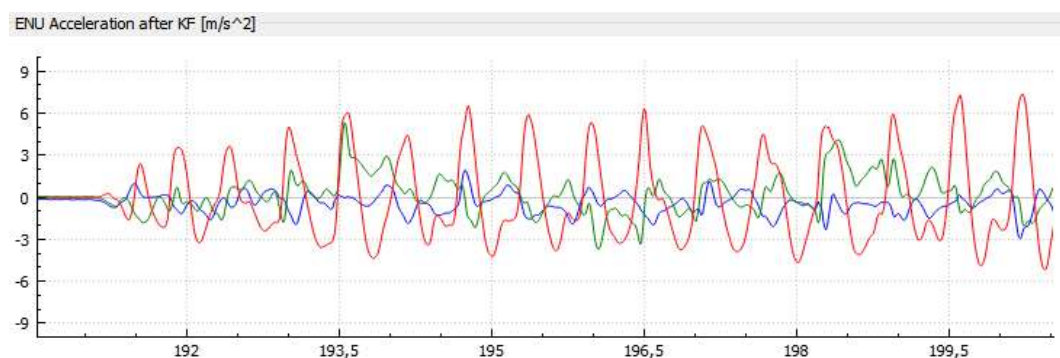


Figure E.9 ENU (linear) acceleration after KF [m·s⁻²] in time [s].

APPENDIX E

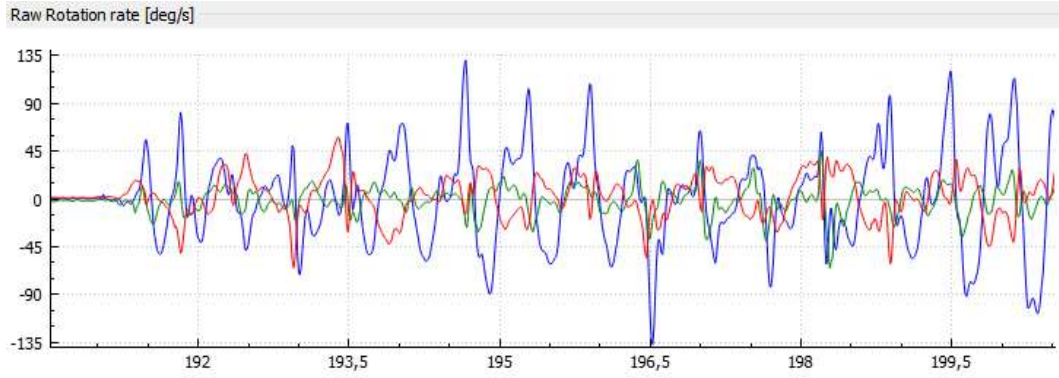


Figure E.10 Raw rotation rate [deg·s⁻¹] in time [s].

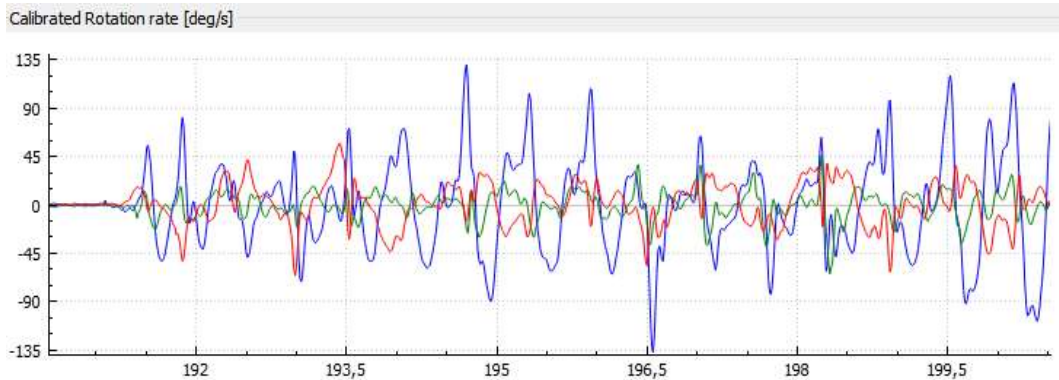


Figure E.11 Calibrated rotation rate [deg·s⁻¹] in time [s].

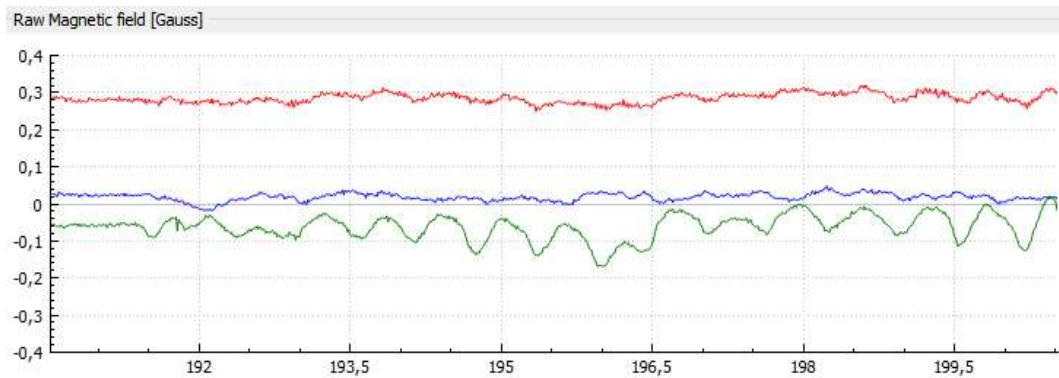


Figure E.12 Raw magnetic field [gauss] in time [s].

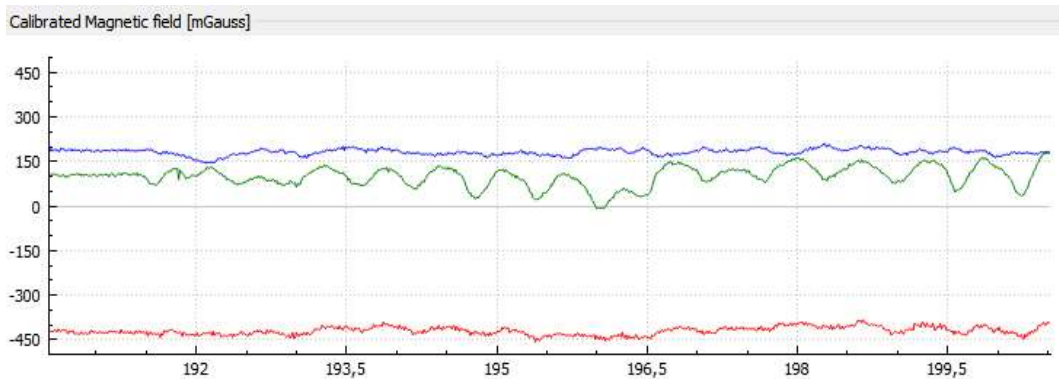


Figure E.13 Calibrated magnetic field [gauss] in time [s].

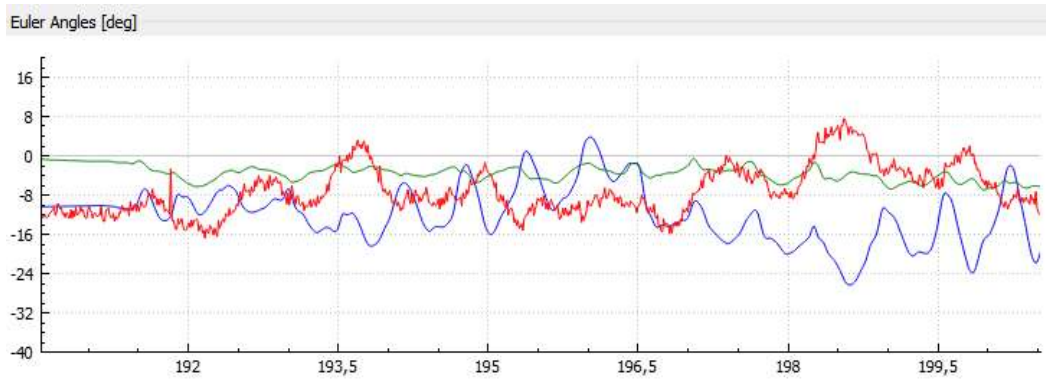


Figure E.14 Euler angles [deg] in time [s].

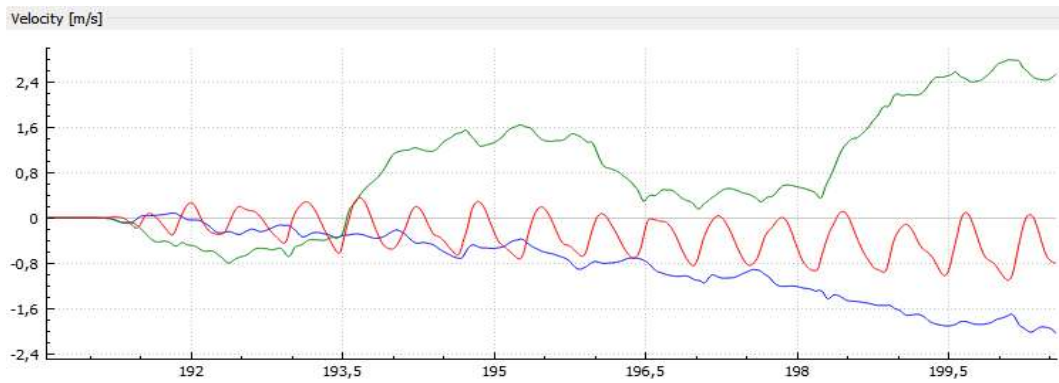


Figure E.15 Velocity [$m \cdot s^{-1}$] in time [s].

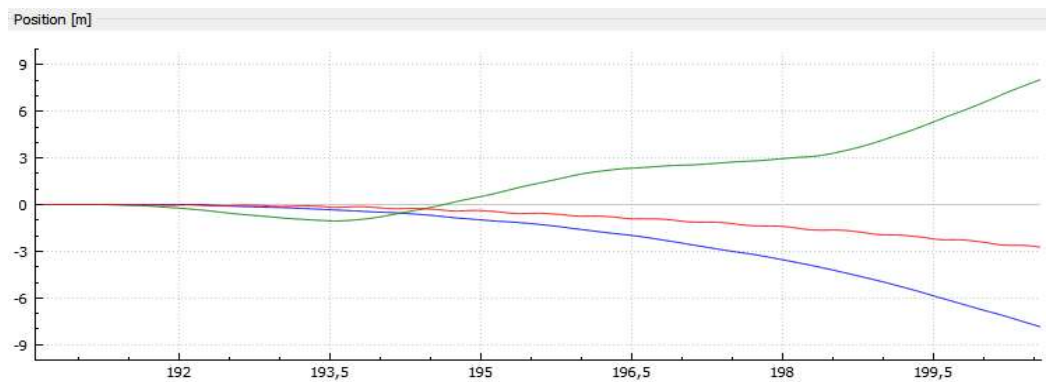


Figure E.16 Position [m] in time [s].

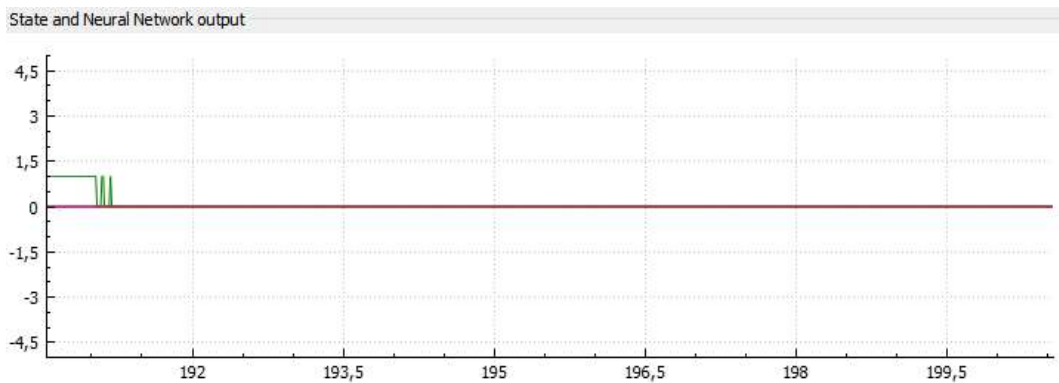


Figure E.17 State and artificial neural network output in time [s].

REFERENCES

- [1] Titterton, D. H. and J. L. Weston. *Strapdown inertial navigation technology*. 2nd ed. Stevenage: Institution of Electrical Engineers, c2004. ISBN 08-634-1358-7.
- [2] Woodman, O. *An introduction to inertial navigation*. Technical Report 696, University of Cambridge, 2007. UCAM-CL-TR-696. ISSN 1476-2986.
- [3] Takai, M. and T. Ura. A model based self-diagnosis system for autonomous underwater vehicles using artificial neural networks. *Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. Tokyo, Japan: IEEE, 1997, p. 82. DOI: 10.1109. ISBN 0-7803-4080-9.
- [4] Nordlund, P.-J. and F. Gustafsson. Recursive estimation of three-dimensional aircraft position using terrain-aided positioning. In: *IEEE International Conference on Acoustics Speech and Signal Processing*. IEEE, 2002, II-1121-II-1124. Orlando, 2002. DOI: 10.1109/ICASSP.2002.5743996. ISBN 0-7803-7402-9.
- [5] Chiang, K.-W. and N. El-Sheimy. INS/GPS Integration Using Neural Networks for Land Vehicle Navigation Application. *Proceedings of the 15th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 2002)*, pp. 535-544. Portland, Oregon, 2002.
- [6] Chiang, K.-W.; El-Sheimy, N.; Lachapelle, G. An Adaptive Neuro-fuzzy Model for Bridging GPS Outages in MEMS-IMU/GPS Land Vehicle Navigation. *Proceedings of the 17th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2004)*, pp. 1088-1095. Long Beach, California, 2004.
- [7] Toth, C.; Grejner-Brzezinska, D.A.; Moafipoor, S. Pedestrian Tracking and Navigation Using Neural Networks and Fuzzy Logic. *2007 IEEE International Symposium on Intelligent Signal Processing*, Alcalá de Henares, 2007, pp. 1-6. DOI: 10.1109/WISP.2007.4447525.
- [8] Moafipoor, S.; Grejner-Brzezinska, D.A.; Toth, C.K. Multi-sensor personal navigator supported by adaptive knowledge based system: Performance assessment. *2008 IEEE/ION Position, Location and Navigation Symposium*, Monterey, CA, 2008, pp. 129-140. DOI: 10.1109/PLANS.2008.4570049.
- [9] Zheping, Y.; Dongnan, Ch.; Zhi, Z.; Chao, D. Dead reckoning error compensation algorithm of AUV based on SVM. *OCEANS 2011 MTS/IEEE KONA*, pp.1-7, Sept. 2011. Waikoloa, HI, 2011. DOI: 10.23919/OCEANS.2011.6107037.
- [10] B. Shin *et al.* Indoor 3D pedestrian tracking algorithm based on PDR using smarthphone. *2012 12th International Conference on Control, Automation and Systems*. JeJu Island, 2012, pp. 1442-1445. 17-21 Oct. 2012. ISBN: 978-89-93215-04-5.
- [11] Wind frame and body-fixed frames. ECI-ECEF and geodetic coordinate frames - Scientific Figure on ResearchGate. [accessed Nov 11, 2015]. Available from: http://www.researchgate.net/figure/278680835_fig1_Figure-4-4.4-ECI-ECEF-and-geodetic-coordinate-frames.
- [12] Wind frame and body-fixed frames. Definition-of-the-body-fixed-frame-with-respec-to-LTP-frame - Scientific Figure on ResearchGate. [accessed Nov 11, 2015]. Available from: http://www.researchgate.net/figure/278680835_fig7_Figure-6-4.6-Definition-of-the-body-fixed-frame-with-respec-to-LTP-frame.
- [13] Grewal, M. S.; Andrews, A. P.; Bartone, Ch. G. *Global navigation satellite systems, inertial navigation, and integration*. Third edition. Hoboken: John Wiley, 2013. ISBN 978-1-118-44700-0.
- [14] Aggarwal, P.; Syed, Z.; Niu, X.; El-Sheimy, N. A Standard Testing and Calibration Procedure for Low Cost MEMS Inertial Sensors and Units. 2008. *Journal of Navigation*, 61, pp 323-336. DOI: 10.1017/S0373463307004560.
- [15] Hamilton, W. R. and Ch. J. Joly. *Elements of quaternions*. [3d ed.]. New York: Chelsea Pub. Co, 1969. ISBN 08-284-0219-1.

-
- [16] Altmann, S., L.; Andrews, A., P.; Bartone, Ch. *Rotations, quaternions, and double groups*. New York: [Ny] udg. Mineola, New York: Dover, 2005. ISBN 04-864-4518-6.
- [17] Yang, Y. Spacecraft attitude determination and control: Quaternion based method. *In Annual Reviews in Control*. Volume 36, Issue 2, 2012, Pages 198-219. ISSN 1367-5788.
- [18] Kuipers, J., B. *Quaternions and rotation sequences: a primer with applications to orbits, aerospace, and virtual reality*. Princeton: Princeton University Press, 2002. ISBN 978-0691102986.
- [19] Hassoun, M. H. *Fundamentals of artificial neural networks*. Cambridge, Mass.: MIT Press, c1995, 511 p. ISBN 02-620-8239-X.
- [20] Tanikic, D. and V. Despotovic. *Artificial Intelligence Techniques for Modelling of Temperature in the Metal Cutting Process*. Metallurgy - Advances in Materials and Processes. InTech, 2012. DOI: 10.5772/47850. ISBN 978-953-51-0736-1.
- [21] Sumathi, S.; Hamsapriya, T.; Surekha, P. *Evolutionary intelligence: an introduction to theory and applications with Matlab*. Online-Ausg. Berlin: Springer, 2008. ISBN 978-354-0753-827.
- [22] Tejmlová, L. and J. Šebesta. Design of wideband Wilkinson dividers using neural network. *In Proceedings of 23rd International Conference Radioelektronika 2013*. 2013. p. 204 - 208. ISBN 978-1-4673-5517-9.
- [23] Tomás, L.-P., and L. Kaelbling. *Techniques in Artificial Intelligence (SMA 5504)*, lecture 2, Fall 2002. Massachusetts Institute of Technology: MIT OpenCourseWare, MIT Course Number 6.825. <http://ocw.mit.edu> (Accessed 4 Feb, 2015). License: Creative Commons BY-NC-SA.
- [24] Omura, Y.; Funabiki, S.; Tanaka, T. A monocular vision-based position sensor using neural networks for automated vehicle following. *Power Electronics and Drive Systems, 1999. PEDS '99. Proceedings of the IEEE 1999 International Conference on*, 1999, pp. 388-393 vol.1. DOI: 10.1109/PEDS.1999.794594.
- [25] Ji, Y.; Zhang, M.; Liu, G.; Liu, Z. Positions research of agriculture vehicle navigation system based on Radial Basis Function neural network and Particle Swarm Optimization. *2010 Sixth International Conference on Natural Computation*. Yantai, Shandong, 2010, pp. 480-484. DOI: 10.1109/ICNC.2010.5583145.
- [26] Hu, Y.; Xu, J.; Zhong, H.; Wu, Y. Fusion model of vehicle positioning with BP neural network. *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems, 2003*, pp. 643-648 vol.1. DOI: 10.1109/ITSC.2003.1252031.
- [27] Touretzky, D. and K. Laskowski. *Artificial Neural Networks: Neural Networks for Time Series Prediction*. *In Engineering Applications of FPGAs*, pp 117-150. Springer, Cham. ISBN 978-3-319-34113-2.
- [28] Bishop, G. and G. Welch. An introduction to the Kalman filter. *Proceedings of SIGGRAPH 2001 course 8, In Computer Graphics, Annual Conference on Computer Graphics & Interactive Techniques*. Los Angeles, CA, USA. SIGGRAPH 2001 course pack edition, 2001.
- [29] Simon, D. *Optimal state estimation: Kalman, H [infinity] and nonlinear approaches*. Hoboken, N.J.: Wiley-Interscience, 2006. ISBN 9780471708582.
- [30] Maybeck, P. S. The Kalman filter: An introduction to concepts. *Autonomous Robot Vehicles*. I. J. Cox and G. T. Wilfong. New York, Springer-Verlag: 194- 204, 1990. ISBN 0387972404.
- [31] Wu, Z.; Yao, M.; Ma, H.; Jia, W. Improving Accuracy of the Vehicle Attitude Estimation for Low-Cost INS/GPS Integration Aided by the GPS-Measured Course Angle. *In IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 2, pp. 553-564, June 2013. DOI: 10.1109/TITS.2012.2224343.
- [32] Hide, Ch.; Moore, T.; Smith, M. Adaptive Kalman Filtering for Low-cost INS/GPS. *Journal of Navigation*, 56, pp 143-152. DOI:10.1017/S0373463302002151.
- [33] Asada, H. *Identification, Estimation, and Learning*. Spring 2006, Massachusetts Institute of Technology: MIT OpenCourseWare, MIT Course Number 2.160. <http://ocw.mit.edu> (Accessed 4 Feb, 2015). License: Creative Commons BY-NC-SA.
-

-
- [34] Sukkarieh, S.; Nebot, E. M.; Durrant-Whyte, H. F. A high integrity IMU/GPS navigation loop for autonomous land vehicle applications. In *IEEE Transactions on Robotics and Automation*, vol. 15, no. 3, pp. 572-578, Jun 1999. DOI: 10.1109/70.768189.
- [35] Tejmlová, L. *Fusion methods for GNSS/INS using neural networks for precision navigation*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2013. 24 p. Dissertation preview. Supervisor: doc. Ing. Jiří Šebesta, Ph.D.
- [36] Barrios, C. and Y. Motai. Improving Estimation of Vehicle's Trajectory Using the Latest Global Positioning System With Kalman Filtering. In *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 12, pp. 3747-3755, Dec. 2011. DOI: 10.1109/TIM.2011.2147670.
- [37] Borenstein, J.; Everett, H. R.; Feng, L.; Wehe, D. *Mobile robot positioning: Sensors and techniques*. *J. Robotic Syst.*, 14: 231-249, 1997. DOI: 10.1002/(SICI)1097-4563(199704)14:4<231::AID-ROB2>3.0.CO;2-R.
- [38] Homolka M. *Inerciální navigační systém*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav radioelektroniky, 2012. 43 p. Bachelor thesis. Supervisor : Ing. Lenka Tejmlová.
- [39] Tejmlová, L. and J. Šebesta. Fusion methods for INS using neural networks for precision navigation. In *International Conference on Indoor Positioning and Indoor Navigation*, 2013, Montbéliard, France. Piscataway, N.J.: IEEE, p. 1-5. ISSN 2162-7347.
- [40] Sheela, K. G. and S. N. Deepa. Review on Methods to Fix Number of Hidden Neurons in Neural Networks. *Mathematical Problems in Engineering*, vol. 2013, Article ID 425740, 11 pages, 2013. doi:10.1155/2013/425740.
- [41] Design Time Series NARX Feedback Neural Networks. *MathWorks: Accelerating the pace of engineering and science*, [online]. 2016 [cit. 2016-07-26]. Available from: <https://www.mathworks.com/help/nnet/ug/design-time-series-narx-feedback-neural-networks.html>
- [42] Time delay neural network. *MathWorks: Accelerating the pace of engineering and science*, [online]. 2016 [cit. 2016-07-26]. Available from: <https://www.mathworks.com/help/nnet/ref/timedelaynet.html>.
- [43] Confusion matrix. *MathWorks: Accelerating the pace of engineering and science*, [online]. 2016 [cit. 2016-07-26]. Available from: <https://www.mathworks.com/help/stats/confusionmat.html>.
- [44] Henke, D. *Magnetometer Reading to Compass Heading*, [online]. Small Golden SceptreTechnology, 2016 [cit. 2016-07-26]. Available from: <http://mythopoeic.org/magnetometer/>.
- [45] NOAA. National Geophysical Data Center: National Centers for Environmental Information [online]. [cit. 2015-4-16]. Available from <http://www.ngdc.noaa.gov/geomag-web>.
- [46] Pololu. MinIMU-9 v2 Gyro, Accelerometer, and Compass. Pololu Robotics and Electronics, [online]. USA, 2013 [cit. 2015-11-09]. Available from: <https://www.pololu.com/product/1268>.
- [47] Calculation of Magnetic Declination. Find Magnetic Declination with Google Maps, [online]. 2015. [cit. 2015-4-16]. Available from: <http://geosats.com/magdecli.html>.
- [48] International Earth Rotation Service (IERS) Service International de la Rotation Terrestre. Federal Agency for Cartography and Geodesy. *Paris Observatory IERS Centers* [online]. [cit. 2016-04-04]. Available from: <http://hpiers.obspm.fr/eoppc/bul/bulb/BULLETINB.html>.
- [49] Kong, X. *Inertial navigation system algorithms for low cost IMU*. University of Technology Sydney, School of Computing and Communications, 2000, 178 p. Doctoral thesis.
-

OWN PUBLICATIONS

Tejmlová, L.; Šebesta, J.; Zelina, P. Artificial Neural Networks in an Inertial Measurement Unit. In *Proceedings of 26th International Conference Radioelektronika 2016*. 2016. p. 176 - 180. ISBN 978-1-5090-1673-0.

Tejmlová, L. and J. Šebesta. Fusion methods for INS using neural networks for precision navigation. In *International Conference on Indoor Positioning and Indoor Navigation. 2013*. p. 814 - 815. ISBN 978-1-4673-1954-6.

Tejmlová, L. and J. Šebesta. Design of wideband Wilkinson dividers using neural network. In *Proceedings of 23rd International Conference Radioelektronika 2013*. 2013. p. 204 - 208. ISBN 978-1-4673-5517-9.

Tejmlová, L. *Ultra-Wide Band Power Divider for Mobile Frequency Bands*. ElectroScope – Online magazine (www.electroscope.zcu.cz), [online], 2012,(5 p.). ISSN~1802-4564.

Tejmlová, L. Ultra-Wide Band Pulse Generator and Positioning System. In *Proceedings of the conference Vsacký Cáb 2012*. 2012. s. 1-6. ISBN: 978-80-214-4579- 6.

Zelinová, L. UWB generátor a system pro určení polohy. Elektrorevue – Online magazine (<http://www.elektrorevue.cz>), [online], 2012, č. 21, s. 1-5. ISSN: 1213- 1539.

CURRICULUM VITAE

PERSONAL DETAILS

Name **Lenka Tejmlová Ing., b.
Zelinová**
Date of Birth **24.9.1986**
Address **Uzbecká 10, 625 00, Brno**
Telephone **+420702089559**
E-mail **tejmlova.lenka@gmail.com**
Nationality **Czech**
Field **Electrotechnics and
communications**



PROFESSIONAL EXPERIENCE

from August 2016 SAP ČR, Vyskočilova 1481/4, Praha 4 – Michle, 140 00
Software developer, department of AIS

April 2013 – Dec. 2014 Dept. of Radioelectronics, Brno University of Technology
Technická 12, 61600 Brno, Czech Republic
Research into wireless channels for intra-vehicle
communication and positioning (GACR no 13-38735S)
Christoph Mecklenbrauker, Ales Prokes

Job description: Publications on current research (VaV)
- UWB technologies, ultra wideband power dividers
- cooperation with Škoda auto a.s.
- ANNs for precision tracking systems (Doctoral Thesis)
- Lessons and leadership (bachelor's and master's theses)

Feb. 2012 – Nov. 2013 ŠKODA AUTO a.s., Tř. Václava Klementa 869,
293 60 Mladá Boleslav
R&D – SoL device, in-car implementation

Oct. 2009 – July 2011 ABB, Vídeňská 117,619 00 Brno, CZ
Switchgear Design, working in a team
Electrical Engineer

June 2007 – Oct. 2007 Giannos Rhodopoulos,
June 2008 – Oct. 2008 Adrina Beach hotel, Skopelos, Greece
June 2009 – Oct. 2009 Electrical maintenance services
Network management and technical support

Feb. 2007 – May 2007 Honeywell, Technická 13, 616 00 Brno, CZ
Research and Development
PCB for the testing of pressure sensor

June 2006 – Nov. 2006 Mediaservis s.r.o, Moravské nám. 13, 602 00 Brno, CZ
Active telemarketing
Operator, communication with customers

EDUCATION

- Sept. 2011 – Sept. 2017** Brno University of Technology, Antonínská 1, 601 90 Brno
Faculty: Faculty of Electrical Engineering and Communication
Program title: Electrical Engineering and Communication
Department of Radio Electronics
Study level: Doctoral, PhD.
Study form: full-time study
- Sept. 2009 – June 2011** Brno University of Technology, Antonínská 1, 601 90 Brno
Study level: Master's, Master's degree, Ing.
- Sept. 2006 – June 2009** Brno University of Technology, Antonínská 1, 601 90 Brno
Study level: Bachelor's, Bachelor's degree, Bc.

SKILLS

LANGUAGE SKILLS:

- NATIVE LANGUAGE: Czech
- LANGUAGES: English - upper-intermediate/advanced
German - basic
Russian - basic
Greek - basic

DRIVER'S LICENSE: B (January 2005) – daily active

TECHNICAL SKILLS: Matlab™, Qt (C-code)
– own source codes, application development
Orcad PSpice, HFSS, CST studio suite
– hardware development, HW testing
CAD, Eplan, Cadelec
– switchgear designing
MS Office

CHARACTERISTICS: reliable, organized, self-reliant, cooperative and hardworking
punctual and accurate
willing to learn, fast learner

ABILITIES: analytical thinking, good communication skills
ability to work in a team as well as self-sufficiently
excellent planning and organising abilities
advanced time management skills
creation of presentations, documentations, articles etc.

INTERESTS: healthy lifestyle
fantasy books
TV documentaries (Discovery, National Geographic)
board games
culture