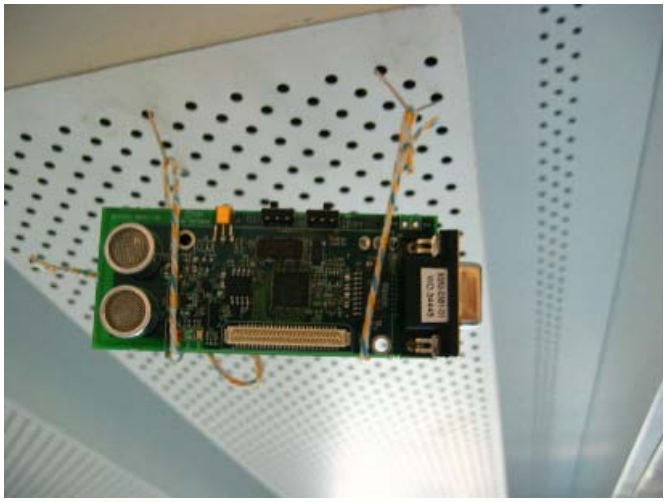
Cricket

Software User Guide



Prepared by: Dr Rainer Mautz

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
• 1.0	• 4.0	• 05.03.2010

Contents

1 Overview 4

1.1 Hardware and Software Requirements 4

1.2 Purpose of the Software 4

1.3 High level architecture overview 4

2 Data Management 5

2.1 The file system 5

2.2 The input data 5

2.3 The output files 6

2.4 The control parameters for the positioning algorithm 6

2.5 Other parameters 7

2.6 Data Structures 8

2.6.1 The Position Format – (e.g. track.out file) 8

2.6.2 The Range Format (e.g. ranges.obs file) 9

3 Functions and Subroutines 10

3.1.1 Main Function ‘cricket.m’ 10

3.2 Functions for Testmode 11

3.2.1 Function ‘load_ranges’ 11

3.2.2 Function ‘getpointset’ 11

3.3 Auxiliary Functions 12

3.3.1 Function ‘rangematrix’ 12

3.3.2 Function ‘connectionsvector’ 12

3.3.3 Function ‘saveranges_crick’ 12
3.3.4 Function ‘savepoints_crick’ 13
3.3.5 Functions ‘cayley_menger2-4’ 13
3.3.6 Function ‘plotcrickets’ 14
3.3.7 Function ‘plotnetwork’ 15
3.3.8 Function ‘plotcluster’ 15
3.3.9 Function ‘mds’ 16
3.3.10 Function ‘tidyP’ 17
3.3.11 Function ‘outlier_detection’ 17

3.4 Functions for Clusterisation 18
3.4.1 Function ‘generate_clusters’ 18
3.4.2 Function ‘merge_clusters’ 18
3.4.3 Function ‘find2clusters’ 18
3.4.4 Function ‘analyse_clusters’ 19
3.4.5 Functions ‘find4’ & ‘find3’ 20
3.4.6 Functions ‘getnext4’ & ‘getnext3’ 20

3.5 Functions for Adjustment 21
3.5.1 Function ‘free_adjust’ 21
3.5.2 Function ‘const_adjust’ 22
3.5.3 Function ‘coarse_positioning 22
3.5.4 Function ‘rotate_system 23
3.5.5 Function ‘transform6’ 23
3.5.6 Function ‘transform_mirror’ 24

3.6 Functions for Lateration 24
3.6.1 Function ‘trilateration’ 24
3.6.2 Function ‘LS_multilateration’ 25
3.6.3 Function ‘iterative_multilateration’ 25
3.6.4 Function ‘calc_quint’ 26
3.6.5 Function ‘bary_quint’ 26
3.6.6 Function ‘quint_error’ 27

3.7 Functions for Performance Assessment 28
3.7.1 Function ‘compare_results’ 28

3.7.2 Function ‘test_orgdata’.....30

4 Components Configuration Description.....31

5 References.....31

6 Directory.....31

6.1 Acronym List.....31

1 Overview

1.1 Hardware and Software Requirements

The positioning software runs on any PC under MATLAB Version 6.5 or higher. A strict requirement on the hardware (memory and disk space) does not exist. The maximum size of the network may be limited according to available internal memory and disk space.

1.2 Purpose of the Software

The program has the capability for 3 dimensional network positioning. The two main functions of the software are the capability of reading ranging data between two Crickets with a ranging capability and the determination of the node positions based on ranges from multiple nodes.

1.3 High level architecture overview

The core of the software package comprises the determination of positions and their related error variances. The principle method to calculate unknown positions is performed by lateration. More precisely, if a surplus of ranges to an unknown device is available, its position is determined by multilateration; otherwise a coarse positioning algorithm based on connectivity information is used. The refinement of the coordinates is realized by a geodetic network adjustment, which is based on the Newton-Raphson gradient method.

The software allows assessment of the positioning performance in the case of the following scenarios and challenges in a positioning network:

- absence or low availability of anchor nodes
- bad geometry of the anchor nodes
- large size of the network
- high variance (random errors) in range measurements
- outliers in the range measurements
- low connectivity (density) of the network
- high requirements on positioning integrity (using an integrity flag)

The following table lists the key localization problems and the implemented methods that deal with them.

Problem	Solution
Absence of anchor nodes	Definition of a local coordinate system, imple-

	mentation of an anchor free localization algorithm, free network adjustment, output of local coordinates.
Low availability of anchor nodes	Implementation of collaborative multilateration clusterisation and merging of clusters.
Large network size	Implementation of a cluster-based localization algorithm.
High variance (random errors) in range measurements	Use of a novel robust lateration algorithm that avoids folding ambiguities and provides good approximate positions for network adjustment.
Outliers in the range measurements	Implementation of data snooping based on a Baarda-Test.
Low connectivity (density) of the network	Implementation of collaborative multilateration, additionally a coarse positioning mode for less connected nodes.

Additionally, there is a graphical display of the created network and the restored network as well as the error vector field between them.

2 Data Management

2.1 The file system

The software is based on text files using the MATLAB m-file format. Besides the main program 'cricket.m' the software consists of several subroutines in the m-file format, totalling 118kB of MATLAB code. The m-files are ASCII files that can be edited in any standard text editor. For a network simulation the input data files are not required a priory, because all data files can be created automatically with this software package.

All data files are text files (ASCII format). Data values are separated by a space (ASCII 32). Each data record ends with the 'new line' character.

2.2 The input data

The input data comes from the cricket listener, via RS232 cable, USB cable or WLAN connection. A typical cricket message (chirp) would be:

```
VR=2.0,ID=01:20:f2:f2:0a:00:00:39,SP=BEACON-5,DB=197,
DR=5764,TM=6314,TS=100512,PC=(0,0,0),TP=20
```

where

VR is the Software Version Running on the beacon cricket node
ID is the unique identification number of the beacon cricket which cannot be changed
SP is the identification of the beacon as given by the user
DB is the range measurement in [cm] (if not set to inches)
DR is the duration [ms] (time of flight of the ultrasound between beacon and listener)
TM is the uncorrected time of flight [ms] (currently not used)
PC are the position coordinates of the beacon cricket (currently all set to 0,0,0). Is not used!
TP is the temperature in [°Celsius] at the beacon node (if not set to Fahrenheit)

More details about the format can be found in the Cricket manual:
http://nms.csail.mit.edu/cricket

Alternatively, in the testmode (testmode = 1), the ranges.obs file is used to read range data. The data file uses the range format 'R'.

2.3 The output files

If not in testmode, two files are written on the hard disk

1) ranges.obs – contains the list of range measurements in the range format 'R'

2) track.out – contains the determined node positions. The format is the positioning format 'P'.

2.4 The control parameters for the positioning algorithm

The parameters regarding positioning are defined in the top of the text file cricket.m, where the values of these parameters are also defined. The following table specifies these parameters, which can be changed by the user in the range as given in the table.

Positioning Parameters				
Parameter Name	Data Type	Value Range	Default	Explanation
iter_max	INTEGER	[1; +∞]	10	maximal number of iterations for the adjustment algorithm
iter_max_multilat	INTEGER	[1; +∞]	10	maximal number of iterations for the multilateration algorithm
precision_x	REAL [m]	[+0; +1]	0.000001	level of numerical precision
sig1	REAL [m]	[0; +10]	2.0	Confidence limit for hypothesis acceptation (e.g.sig1 = 2 ➔ 0.9545)

sig2	REAL [m]	[0; +10]	3.0	Confidence limit for hypothesis rejection (e.g.sig1 = 3 ➔ 0.99730)
use_multiit	BOOLEAN	0, 1	1	Uses novel multilateration (=1) or conventional multilateration (=0).
use_volume	BOOLEAN	0, 1	1	Performs volume integrity check (=1) or omits this step (=0).
use_ambi	BOOLEAN	0, 1	1	Performs statistical ambiguity check (=1) or omits this step (=0).

2.5 Other parameters

The remaining parameters allow control of other aspects than positioning and simulation.

Positioning Parameters				
Parameter Name	Data Type	Value Range	Default	Explanation
plotting	BOOLEAN	0, 1	1	Plots the location of the listener (= 1) or omits it (= 0).
font	INTEGER	[1; 50]	15	value for the size of the fonts in all graphic output
rangefile	STRING		ranges.obs	name of file with raging data
trackfile	STRING		track.out	name of file with position data
loops	INTEGER	[1; +∞]	200	number of collected ranges until program stops
maxerror	REAL [m]	[0; +∞]	0.10	maximal position error that allows plotting of the point
timeout	REAL [s]	[0; +∞]	1.5	time how long a range measurement should be valid for position computation
sig02, sig1, sig2, apriory_variance, ppm				currently not in use

2.6 Data Structures

The data formats are of struct-type (containing several hybrid data types) and include: position format (P), range format (R).

2.6.1 The Position Format – (e.g. track.out file)

The Position Format is set up to store all node related data.

Structure name: P

Data	Symbol	Data Type	Value Range	Example
Device ID	ID	integer	[1; +∞]	0
Time Stamp [s]	t	REAL []	[0; +∞]	5.844
Maximal Range [m]	r_{max}	REAL	[0; +∞]	10.00
Status	S	INTEGER	[0, 1, 2, 3]	0 not used
Position availability	A_v	INTEGER	[0, 1, 2, 3]	2
X-Coordinate [m]	X	REAL	[−∞; +∞]	7.596
Y-Coordinate [m]	Y	REAL	[−∞; +∞]	3.793
Z-Coordinate [m]	Z	REAL	[−∞; +∞]	-0.569
Variance in X [m]	σ_x	REAL	[−∞; +∞]	0.0226
Variance in Y [m]	σ_y	REAL	[−∞; +∞]	0.0216
Variance in Z [m]	σ_z	REAL	[−∞; +∞]	0.0236
Point Variance [m]	σ	REAL	[−∞; +∞]	0.0392
Reliability	Re	INTEGER	[0, 1, 2]	6
Internal Point Number*	p	INTEGER	[0, 1,…, n]	4
Variance-Covariance Matrix*	Q	3 x 3 matrix of REAL		$\begin{bmatrix} 2.4 & 7.5 & 0.3 \\ 0.6 & 1.0 & 0.1 \\ 0.4 & 0.5 & 0.9 \end{bmatrix}$

* does not appear in the data file (track.out)

The following remarks aim to clarify some of the data fields in the table above:

The '**SIP ID'** and the **'status' S** are currently not in use.

The 'Device ID' can be any integer number, but must be unique.

The time stamp 't' is the time in seconds, when the range measurement arrived at the server.

The **'position availability' A_v** can take the values -1, 0, 1, 2 or 3.

-1 = Position information is not available due to a singular matrix

0 = Position information is not available (has not yet been calculated)

1 = Rough approximate position information is available (calculated by 3 ranges)

2 = Position information is available (determined redundantly)

3 = Position information is available in the reference system (anchor node)

In case the 'position availability' A_v = -1 or 0 all further data in the P-file are not expected and will be ignored if nevertheless provided.

The **'reliability' Re** can take the values 0, 3, 4,..n and gives the number of ranges used to determine the node position by multilateration.

3 = no redundancy. Position is not reliable. Due to outliers, the position may erroneous beyond the variance information.

4 = some redundancy. The position is verified due to one redundant measurement, but there is not enough reliability that fulfils strict integrity requirements.

>4 the position is reliable and fulfils the integrity requirements.

2.6.2 The Range Format (e.g. ranges.obs file)

The Range Format is set up to store all the data that is related to a range measurement between two devices. It differs largely from the range-format as coming from the cricket. The order of the value does matter.

Structure name: R

Data	Symbol	Data Type	Value Range	Example
SIP ID*	SIP	CHAR []	email address	rm@ethz.ch
Device ID 1	ID1	INTEGER	[0, 1,…, n]	0
Device ID 2	ID2	INTEGER	[0, 1,…, n]	5

Time stamp [s]	t	REAL	[0; +∞]	5.844
RSSI [dB]	RSSI	REAL	[−∞; +∞]	5
Range availability	av	BOOLEAN	0, 1	1
Range [m]	r	REAL	[0; +∞]	1.35
Mean error [m]	m	REAL	[0; +∞]	0.02
Weight [m]	w	REAL	[0; +∞]	0.85
Internal number*	n1	INTEGER	[0, 1,…, n]	1
Internal number*	n2	INTEGER	[0, 1,…, n]	4

* does not appear in the range file (ranges.obs)

The '**SIP ID', RSSI, 'Mean error' m,** and the **'weight' w** are currently not in use. If a value is displayed nevertheless, it is an arbitrary constant.

The **'range availability' av** has the following options:

0 = no range could be measured, but there is a data link. In this case, the range data field is expected to be empty

1 = range could be measured

3 Functions and Subroutines

3.1.1 Main Function 'cricket.m'

'cricket.m' opens an instrument (either connected to serial or USB port). It expects ranging data from a cricket listener coming in continuously. The cricket listener should be configured with P CO *, so that all parameters are transferred to the host where cricket.m is running. Cricket.m reads than from the text strings the values for SP, DB, DR, TM, TS, TP (for details see Cricket manual). If SP (the identifier of a cricket) has not been there before, a new point P is created (without any position information yet). If a node has known coordinates (which have been directly entered in the cricket.m file), its coordinates are loaded. The range measurements are stored using the ranging format R and recent ranges are stored in the variable L. Based on the up-to-date ranges L and the anchor node Positions P(2..n), the mobile Position P(1) is determined. If there are only 3 available ranges, the function trilateration is used, if there are more than 3 ranges, the function LS_multilateration (least squares multilateration) is used. Note that LS_multilateration determines the rover position redundantly and computes error information based on the residuals.

3.2 Functions for Testmode

The program runs in testmode, if the 'testmode' parameter is set to the value 1. If the software runs in testmode, no cricket hardware needs to be connected to the PC. The main purpose for the testmode functionality is to test the positioning software without the need for ranging data from a cricket. While running in testmode, the ranging data comes from a file on disk (usually ranges.obs).

3.2.1 Function 'load_ranges'

The function call 'load_ranges' is an alternative to obtaining range measurements directly from Cricket devices if the ranging data is to be read from an existing file. This function is useful to make multiple runs of the software with the same ranging data.

Function name:	LOAD_RANGES
Function call:	R = load_ranges('ranges100.obs')
Input:	range_in: name of a range file
Output:	R: Range-Structure with values
Description:	opens and reads the ranging data from the specified file. The data is stored and returned in the structure R.

3.2.2 Function 'getpointset'

The function call 'getpointset' creates a list of point- (or device-) numbers based on the incoming range information R. A point list P is returned with all the point-id numbers that appear in the ranges. The coordinates of the points are all 0 as well as the 'availability' av.

Function name:	GETPOINTSET
Function call:	[P, R] = getpointset(R)
Input:	R: Range-Structure with ranging information
Output:	P: List of points (nodes), that have ranges to other nodes R: Updated Range-Structure
Description:	finds all point-id numbers that appear in the range observations.

3.3 Auxiliary Functions

3.3.1 Function 'rangematrix'

The function 'rangematrix' uses a list of ranges R to set up an *m* x *m* matrix where *m* is the number of ranges. Zero values indicate that there is no range available. The transformation of the ranges into matrix form simplifies various mathematical operations on the data.

Function name:	RANGEMATRIX
Function call:	C = rangematrix(R)
Input:	R: Range-Structure with ranging information
Output:	C: *m* x *m* matrix with range values
Description:	Transforms all available ranges into a matrix form.

3.3.2 Function 'connectionsvector'

The function 'connectionsvector' uses a list of ranges R to set up a vector with the number of connections for each node (=each device). The order of the nodes is controlled by the internal point nubers.

Function name:	CONNECTIONSVECTOR
Function call:	cv = connectionsvector(R)
Input:	R: Range-Structure with ranging information
Output:	cv: vector with *m* elements and connections counts.
Description:	Counts the number of connections for each node.

3.3.3 Function 'saveranges_crick'

The function 'saveranges_crick' takes a list of ranges and writes all the relevant data onto hard disc using the point format R. The filename is specified as a parameter. The SIP is not written. Note that there is no header.

Function name:	SAVEPOINTS
Function call:	savepoints(filename, R)
Input:	filename: name of a non-existing file or an existing file that can be overwritten R: List of ranges, see 2.6.2
Output:	none.
Description:	Writes range related data in a specified file.

3.3.4 Function 'savepoints_crick'

The function 'savepoints_crick' takes a list of nodes and writes all the relevant data onto hard disc using the point format P. The filename is specified as a parameter. No header is written on top of the new file.

Function name:	SAVEPOINTS
Function call:	savepoints(filename, P)
Input:	filename: name of a non-existing file or an existing file that can be overwritten P: List of nodes and their positions, see 2.6.1
Output:	none.
Description:	Writes position related data in a specified file.

3.3.5 Functions 'cayley_menger2-4'

The function 'cayley_menger4' creates and computes a Cayley-Menger determinant of size 4. It expects a 4 x 4 matrix of squared distances between 4 nodes, where the diagonal elements have zero values.

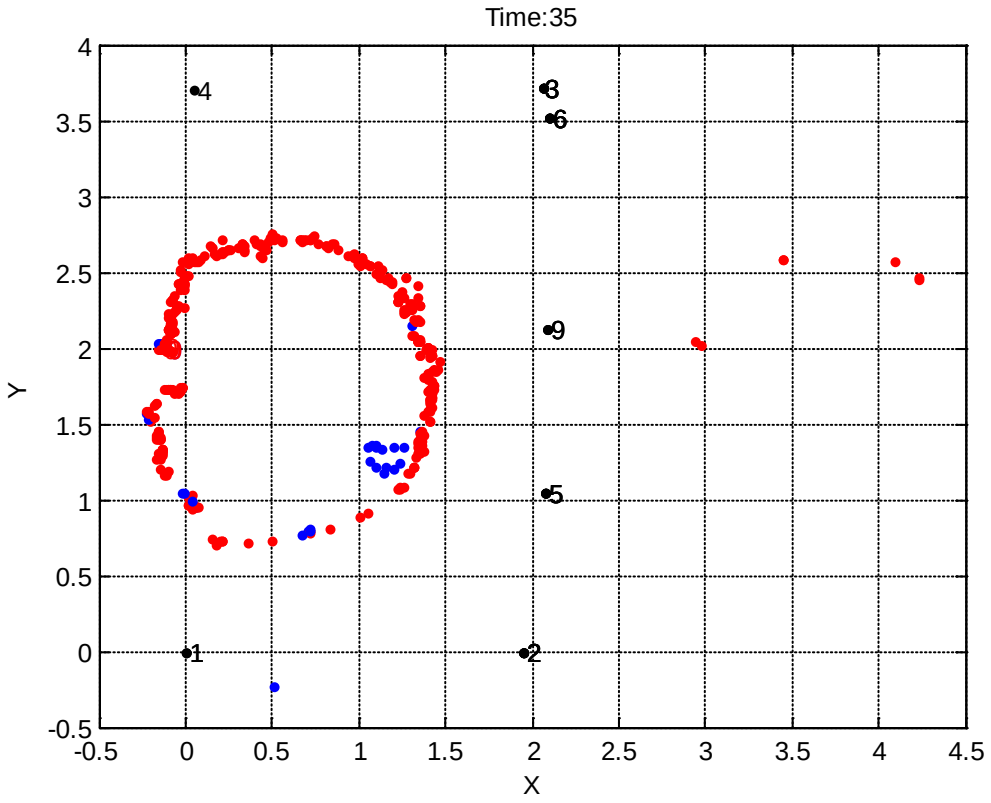
The functions 'cayley_menger3' and 'cayley_menger2' determine the 3 x 3 and 2 x 2 determinants accordingly. For further details on the usage of Cayley-Menger matrices, see Thomas and Ross (2005).

Function name:	CAYLEY_MENGER4
Function call:	d = cayley_menger4(D)
Input:	D: 4 x 4 matrix with squared distances between the 4 nodes
Output:	d: value of Cayley-Menger determinant
Description:	Creates a Cayley-Menger determinant based on given squared ranges and computes it.

3.3.6 Function ‘plotcrickets’

The function ‘plotcrickets’ creates a 2D plot of the nodes specified in the structure ‘P’. The value of this function is to have a quick overview of the geometry and also the movement of nodes. Blue dots denote that the position was estimated based on 3 nodes only; red dots denote that there were more than 3 nodes available for the position computation. Fixed nodes are drawn in black.

Function name:	PLOTCRICKETS
Function call:	plotncrickets(P, t, plotallpoints)
Input:	P: List of nodes and their positions, see 2.6.1
	R: Range-Structure with ranging information, see 2.6.2
	t: string for title – could be the time information
	plotallpoints: 0 = plots only point P(1), 1 = plots all points in P
Output:	none
Description:	Plots a 2D view of the point locations.

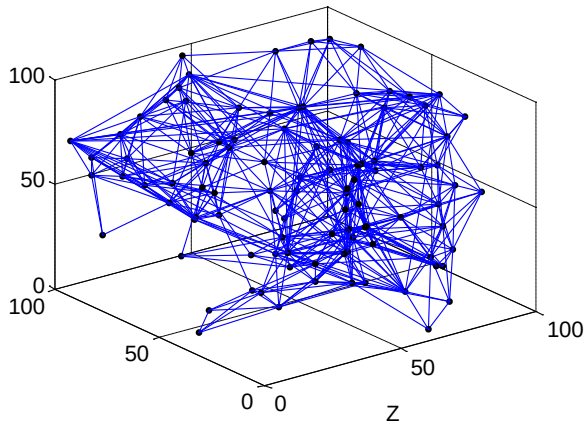


Example plot using the function ‘plotcrickets’

3.3.7 Function ‘plotnetwork’

The function ‘plotnetwork’ creates a 3D plot of the nodes specified in the structure ‘P’ and also plots the given range observations. The value of this function is to have a quick overview of the geometry and also the movement of nodes.

Function name:	PLOTNETWORK
Function call:	plotnetwork(P, R, text)
Input:	P: List of nodes and their positions, see 2.6.1
	R: Range-Structure with ranging information, see 2.6.2
	text: string for title
Output:	none
Description:	Plots a 3D view of the point locations and inter-nodal ranges.

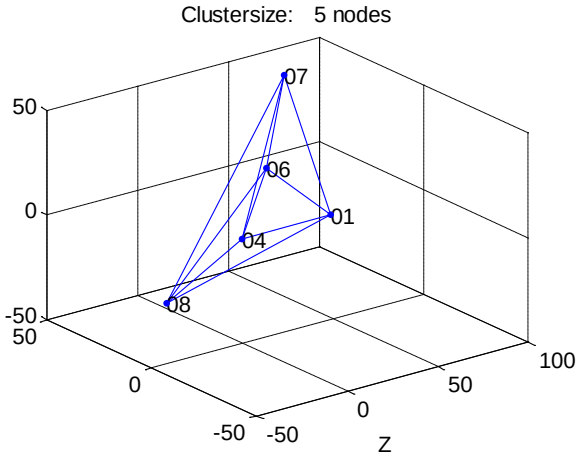


Example plot using the function ‘plotnework’

3.3.8 Function ‘plotcluster’

The function ‘plotcluster’ creates a 3D plot of a cluster of nodes (e.g. a quintilateral), their inter-node ranges and point numbers. The value of this function is to have a quick overview of the geometry and location of the nodes.

Function name:	PLOTCLUSTER
Function call:	plotcluster(CL, C, text)
Input:	CL: One cluster of nodes. See 2.6.3.
	C: rangematrix, see 3.2.1
	text: string for title
Output:	none
Description:	Plots a 3D view of the specified cluster.



Example plot using the function ‘plotcluster’

3.3.9 Function ‘mds’

MDS is the abbreviation for Multi-Dimensional Scaling, which is a common mathematical tool in positioning. MDS is actually not part of the positioning strategy in the software package. It is merely used for comparison, as it is a competitive algorithm.

MDS uses a ranging matrix to determine all node positions of the network. If the matrix with ranges is incomplete (i.e. not all ranges have been observed), it is completed by a scheme that uses the shortest path to estimate the missing links. In a second step, a Singular Value Decomposition (SVD) is carried out. If the cluster is rigid, the result can be meaningful; otherwise the positional solutions are questionable.

Function name:	MDS
Function call:	P = mds(C, S, R)
Input:	C: rangematrix, see 3.2.1 S: matrix with variances of the ranges (error information) R: Range-Structure with ranging information, see 2.6.2
Output:	P: List of nodes and their positions, see 2.6.1
Description:	Determines the positions of the nodes in a network.

3.3.10 Function 'tidyP'

The function 'tidyP' receives a list of nodes P and copies them into a new cluster of nodes. Only those nodes with a position availability Av>0 are copied. Note that the length of the new cluster is the length of PID (which is the list of all ID numbers).

Function name:	TIDYP
Function call:	CL = tidyP(P)
Input:	P: List of nodes, see 2.6.1
Output:	CL: One clusters with available nodes. See 2.6.3
Description:	Finds nodes with available positions and stores them in a new cluster.

3.3.11 Function 'outlier_detection'

The function 'outlier_detection' receives the residual information of an adjustment and performs a Baarda test. In case the Baarda test statistic detects an outlier observation in a set of ranges, the related range measurement is deleted from the dataset. Multiple outliers are detected sequentially.

Function name:	OUTLIER_DETECTION
Function call:	[R, cv, C, S] = outlier_detection(R, v, m0f, C, P)
Input:	R: Range-Structure with ranging information, see 2.6.2 v, m0f: residuals and mean deviation from adjustment C: rangematrix, see 3.2.1 P: List of nodes, see 2.6.1
Output:	R, cv, C, S: Updated ranges R, connectionvector cv, range-matrix C and matrix with variances of the ranges S.
Description:	Finds and deletes single outlier observations.

3.4 Functions for Clusterisation

3.4.1 Function ‘generate_clusters’

The function call ‘generate_clusters‘ is a meta function that calls several other subroutines. Its purpose is to generate rigid clusters of nodes and determine local positions of the participateng nodes.

Function name:	GENERATE_CLUSTERS
Function call:	CL = generate_clusters(cv, C, S)
Input:	cv: connections_vector, see 3.2.2
	C: rangematrix, see 3.2.1
	S: matrix with variances of the ranges (error information)
Output:	CL: List of clusters with well connected nodes. See 2.6.3.
Description:	finds and determines a rigid graph.

3.4.2 Function ‘merge_clusters’

The function call ‘merge_clusters‘ is also a meta function that calls several other subroutines. Its purpose is to find in a larger array of clusters two clusters that share three or more nodes. If two clusters with sufficient common nodes are found, they are merged to one bigger cluster. The two smaller clusters are deleted from the list.

Function name:	MERGE_CLUSTERS
Function call:	CL = merge_clusters(cv, C, S)
Input:	cv: connections_vector, see 3.2.2
	C: rangematrix, see 3.2.1
	S: matrix with variances of the ranges (error information)
Output:	CL: Updated list of clusters, see 2.6.3.
Description:	reduces the number of small clusters in an array of clusters and returns one (or more) larger clusters.

3.4.3 Function ‘find2clusters’

The function call ‘find2clusters’ is an auxiliary function for the process of merging clusters. Out of a list of clusters, the function identifies 2 clusters that share a specified number of common nodes.

Function name:	FIND2CLUSTERS
Function call:	[i, j, c] = find2clusters(CL, co, CLOK)
Input:	CL: list of clusters, see 2.6.3 co: specified minimum number of required common nodes CLOK: matrix that specifies which combinations of clusters have already been tested
Output:	i: cluster number of the bigger cluster j: cluster number with the smaller cluster c: number of common points of the two clusters
Description:	identifies two clusters sharing a specified number of common nodes.

3.4.4 Function 'analyse_clusters'

The function call 'analyse_clusters' assesses the inter-cluster connectivity between the two clusters specified. It analyses all nodes and groups the nodes according to their membership of the two clusters. Range measurements between the two clusters are identified.

Note: Nodes that have no positioning availability (Av < 2), are not considered as a member of a cluster.

Function name:	ANALYSE_CLUSTERS
Function call:	[av0, av1, av2, av3, p1, p2, r] = analyse_clusters(A,B,C)
Input:	A: Cluster of nodes B: Cluster of nodes C: rangematrix, see 3.2.1
Output:	av0: nodes that are neither in A nor in B av1: nodes that are only in A but not in B av2: nodes that are only in B but not in A av3: nodes that are in A and B p1, p2: nodes that have ranges to the other clusters r: ranges between the two clusters (only numbers)
Description:	analyses two specified clusters.

3.4.5 Functions ‘find4’ & ‘find3’

The function call ‘find4‘is a tool that finds four nodes with a complete set of inter-node ranges using the connectivity matrix C. The function supports the problem of finding a rigid structure on the graph of a set with nodes and range measurements between them.

The search is started at a specified set of four nodes. If these four nodes are not connected, i.e. one range out of the required six ranges is missing, the search is continued with the next combination of nodes. If no complete set of ranges can be found, an empty vector is returned.

The function ‘find3’ is identical to ‘find4’ where the only difference is that 3 connected nodes are to be found.

Function name:	FIND4
Function call:	q = find4(C, m)
Input:	C: rangematrix, see 3.2.1 m: vector of 4 INTEGER values that denote the point numbers of 4 nodes, where the search is started from. The requirement is that m[1] < m[2] < m[3] < m[4].
Output:	Number of four nodes with a complete set of inter-node ranges
Description:	finds four nodes that have ranges to reach other.

3.4.6 Functions ‘getnext4’ & ‘getnext3’

The function call ‘getnext4’ supports the search on the connectivity path. It returns the next set of 4 different point numbers in a network. The sequence of point numbers is defined from 1 to *n*, where *n* is the total number of points. Each function evaluation increments the last of the four point numbers by one. If the maximum *n* is reached, the second last digit is incremented.

The function ‘getnext3’ is identical to ‘getnext’ where the only difference is that 3 point-numbers are returned.

Function name:	GETNEXT4
Function call:	q = getnext4(C, m)
Input:	C: rangematrix, see 3.2.1 m: vector of 4 INTEGER values that denote the internal point numbers of 4 nodes, where the requirement is that m[1] < m[2] < m[3] < m[4].
Output:	next set of four nodes in the sequence.
Description:	finds the next set of internal point numbers of four nodes.

3.5 Functions for Adjustment

3.5.1 Function 'free_adjust'

The purpose of this function is a refinement of coordinates (= locations of devices). The adjustment method is free, because no anchor (=reference points) are necessary. The geodetic datum (3 translations, 3 rotations) is the inner constraints datum, where the translation causes the barycentre to be the coordinate origin and the rotation between the approximate coordinates and the adjusted coordinates is minimised.

The function call 'free_adjust' uses a given point field with estimates of the node positions and the inter-node ranges to determine new node positions that have a better fit (in the sense of least squares) of the inconsistencies within the range observations. The algorithm uses the Gauss-Newton optimization, which belongs to the class of gradient methods. The initial position estimates must be already 'close' to the true location, otherwise the algorithm may fail.

Function name:	FREE_ADJUST
Function call:	[P, v, m0f, redundancy] = free_adjust(C, S, P)
Input:	C: rangematrix, see 3.2.1 S: matrix with variances of the ranges (error information) P: List of nodes and their approximate positions, see 2.6.1
Output:	P: Updated list of nodes v: residuals (=inconsistencies after adjustment) m0f: empirical mean deviation after the adjustment redundancy: number of redundant measurements
Description:	minimisation of the residuals using the least-squares criterion uses an inner constraints datum to define a local coordinate system, returns all coordinates in a local system with the origin in the barycentre.

3.5.2 Function ‘const_adjust’

The purpose of this function is a refinement of coordinates. The major difference to the function ‘free_adjust’ is that fixed anchor points in a given reference system are required. The returned coordinates of all nodes are in the geodetic datum of the reference system. At least 3 nodes with coordinates in the reference system are required. The coordinates of these so-called fixed nodes remain unchanged.

Function name:	CONST_ADJUST
Function call:	[P, vc, m0c, redundancy] = const_adjust(R, P)
Input:	R: Range-Structure with ranging information, see 2.6.2 P: List of nodes and their approximate positions, see 2.6.1
Output:	P: Updated list of nodes vc: residuals (=inconsistencies after adjustment) m0c: empirical mean deviation after the adjustment redundancy: number of redundant measurements
Description:	minimisation of the residuals using the least-squares criterion, the geodetic datum is defined by the fixed nodes in the reference system.

3.5.3 Function ‘coarse_positioning

The purpose of this function is to add those nodes to the network, which could not participate in the precise geodetic adjustment. The nodes added to the network by coarse_positioning may have 1, 2, 3 or more ranges to other nodes. Three cases are distinguished and dealt separately: a) one single range, b) two ranges and c) 3 or more ranges. The rough coordinate estimates and their variances are returned.

Function name:	CORSEPOSITIONING
Function call:	P=coarse_positioning(P, R)
Input:	P: List of nodes and their approximate positions, see 2.6.1 R: Range-Structure with ranging information, see 2.6.2
Output:	P: Updated list of nodes
Description:	calculates rough coordinate estimates for nodes P based on 3 or less ranges to known nodes.

3.5.4 Function 'rotate_system

Rotate_system is a simple function that translates and rotates data points. Although the input data is of 3D-coordinate tpye, the z-components are ignored and the rotation is defined around the z-axis only.

Currently, the translation is as follows: point P(1) is origin. The x-axis is in direction to P(2).

Function name:	ROTATE_SYSTEM
Function call:	ROTATE_SYSTEM
Input:	none, but a list of nodes P is written on top of the file,
Output:	none, but list of transformed coordinates P is printed.
Description:	Transforms coordinates to the reference system A, based on a least squares minimisation criterion.

3.5.5 Function 'transform6'

Transform6 is a function that determines the transformation parameters of a 3D-6 parameter transformation (3 translations, 3 rotations). The set of coordinates is transformed after the transformation parameters have been estimated.

The function transform6 receives two sets of coordinates. The first set is presumed to be a set of anchor nodes in the reference system. The second set of coordinates is transformed to the reference system, if there are more than 3 common points in both datasets. For the determination of the transformation parameters, the closed-form solution from Horn (1987) is used. For the absolute orientation orthonormal matrices are set up. With the help of quaternion algebra, the mathematical formulas are simplified.

Function name:	TRANSFORM6
Function call:	[P, svv] = transform6(A, B)
Input:	A: list of anchor nodes, B: list of nodes in a local coordinate system (to be transformed)
Output:	P: list of transformed coordinates. svv: sum of the squared residuals (fit-function assessing the inconsistencies.
Description:	Transforms coordinates to the reference system A, based on a least squares minimisation criterion.

3.5.6 Function 'transform_mirror'

The function transform_mirror receives two sets of coordinates. The first set is presumed to be a set of anchor nodes in the reference system. The second set of coordinates is transformed to the reference system, if there are more than 3 common points in both datasets. The second data set is transformed twice: once using the original data set and then using a mirrored version of the node positions. The results of both transformations are compared and the one with the better fit is returned.

Function name:	TRANSFORM_MIRROR
Function call:	P = transform_mirror(A, B)
Input:	A: list of anchor nodes, B: list of nodes in a local coordinate system (to be transformed)
Output:	P: list of transformed coordinates.
Description:	Transforms the data in B to the reference system A. Tests, whether the original or the mirrored version of B has better fit to A. Returns the transformed coordinates.

3.6 Functions for Lateration

3.6.1 Function 'trilateration'

The trilateration function uses the efficient algorithm developed by Thomas, F and Ros, L (2005). The coordinates of one unknown point in 3D are determined from three known beacon points with ranges to the unknown listener. First, a 4 by 4 range matrix is set up. The calculation of the unknowns is done by using barycentric coordinates and Caylay-Menger matrices. For the general case, two solutions for the node location are returned (both possible embeddings of the graph which are mirrored at the base plane). If there is only an imaginary solution, the point availability 'av' is set to zero. In case of a unique solution, the coordinates are returned in a set of two (in order to keep the format of the general case).

Function name:	TRILATERATION
Function call:	P = trilateration (P, [P_1, P_2, P_3], [r_1, r_2, r_3])
Input:	P: node to be trilaterated, coordinates are overwritten [P_1, P_2, P_3]: Three known nodes (with A_v= 2, 3), [r_1, r_2, r_3]: ranges between the three known nodes and P.
Output:	P with trilaterated coordinates
Description:	calculates coordinates for node P based on 3 anchor points if coordinates of these 3 nodes are available.

3.6.2 Function 'LS_multilateration'

The function LS_multilateration is a complex function which uses multiple other subroutines. The purpose is to determine the coordinates of an unknown node, if more than 3 ranges from other nodes are available. Approximation values for the unknown node are not necessary. The function solves the ambiguity problem and finds the most likely node position out of multiple folding ambiguities. The function also minimises the residuals based on the least-squares criterion. Several integrity checks are integrated in order to rule out folding errors or constellations with bad geometry.

Function name:	LS_MULTILATERATION
Function call:	P = LS_multilateration(P, F, d, s)
Input:	P: unknown node to be laterated, F: [P_1, P_2, …, P_n] (with n ≥ 4 and A_v= 2, 3), 4 known nodes d: vector with 4 or more ranges to P s: vector with the associated mean observation errors
Output:	P coordinates and error variances.
Description:	uses 4 or more ranges to anchor nodes to determine one unknown node.

3.6.3 Function 'iterative_multilateration'

The function iterative_multilateration is also a complex function that calls LS_multilateration. The purpose is to extend an existing rigid cluster (e.g. a quintrilateral consisting of 5 nodes) into a larger cluster. The adding of nodes is done iteratively (one by one) using multilateration. The expansion is continued until no more nodes that fulfil the requirements on multilateration can be found. The cluster is adjusted and can optionally be displayed.

Function name:	ITERATIVE_MULTILATERATION
Function call:	CL = iterative_multilateration(CL, C, S)
Input:	CL: small cluster 5 or more positioned nodes, C: rangematrix, see 3.3.1 S: matrix with variances of the ranges (error information)
Output:	CL: expanded cluster with coordinates and error variances.
Description:	extends a rigid cluster of nodes and determines their coordinate positions.

3.6.4 Function 'calc_quint'

The function calc_quint creates a cluster of 5 nodes, a so-called quintilateral or quint. All 10 range measurements between the 5 nodes are required. They are handed over in the range matrix C. Calc_quint runs a geometry test and if it passes the test, local coordinates are assigned to its vertices. If it fails the test, the point availability is set to -1. The determination of the coordinates is done by trilateration, multilateration followed by an over-determined least-squares adjustment.

Function name:	CALC_QUINT
Function call:	Q = calc_quint (q, C, S)
Input:	q: list of 5 nodes with complete inter-node ranges C: 5 x 5 matrix with inter-node range observations S: 5 x 5 matrix with the corresponding variances.
Output:	Q: Cluster with 5 nodes, containing positional information
Description:	creates a quint (cluster of 5 nodes) and determines local coordinates and variances for the nodes.

3.6.5 Function 'bary_quint'

The function bary_quint determines the barycentric coordinates of a quintilateral (5 nodes with range observations to each other). It also assesses important parameters on the geometry of the quintilateral including the two tetrahedrons that make up a quint. A geometry check is carried out in order to assess the robustness of the quint in the presence of noise. For instance, too flat tetrahedrons are flagged as not robust, if their height is lower than the mean error of the height. Furthermore, both possible embeddings of the quint are statistically assessed and only if one embedding is significant due to hypothesis testing, it is flagged as robust. One of the main criteria to decide on the robustness is the mean error of the redundant range measurement between nodes 4 and 5. For further details, see Mautz et al. (2010).

Function name:	BARY_QUINT
Function call:	[D1, D2, D3, D34, D35, D4, D25, D24, D5, D6, h4, h5, dhq, dhqs, dqq, l45, l45s, dd, dds, m45, m45s, sok, ook] = bary_quint(C, S)
Input:	C: 5 x 5 matrix with inter-node range observations S: 5 x 5 matrix with the corresponding variances
Output:	D1…D6: barycentric coordinates of the two tetrahedrons h4: height of the tetrahedron '1234' h5: height of the tetrahedron '1235' dhq (dhqs): squared height difference when nodes 4 and 5 are on the same (other) side of the base plane 1,2,3 dqq: distance of the foot-points of 4 and 5 in the plane 1,2,3 l45 (l45s): calculated distance between 4 and 5 if on the same (different) side of the base plane 1,2,3 dd (dds): difference between l45 (calculated) and r45 (observed) if on the same (different) side of the base plane 1,2,3 m45: mean error of the difference between l45 calculated and r45 observed. m45s: same as m45, but this time point 5 is on the other side as point 4. sok: 'same side ok', flags 1 if tetrahedron is robust otherwise 0 ook: 'other side ok', flags 1 if reflection is robust, otherwise 0
Description:	determines parameters related to a quint.

3.6.6 Function 'quint_error'

The function 'quint_error' determines the mean error of one inter-nodal distance in a quintilateral. The function is a tool for solving the ambiguity problem (see Mautz et al. 2010) that occurs when a quintilateral is determined. A quintilateral consists of 10 range measurements, with 9 of them being necessary to set up the coordinate system, there is one redundant measurement, here referred as r45. The function determines the mean error of the distance between nodes 4 and 5 without using the actual range measurement r45. The distance between nodes 4 and 5 (here denoted by l45) is determined from the remaining 9 range measurements using barycentric coordinates. The function 'quint_error' does determine the error of the difference between r45 (measured) and l45 (determined) using the strict law of error propagation. The error propagation is carried out exploiting the chain rule and making use of propagating the errors in matrix form throughout all the mathematical formulas.

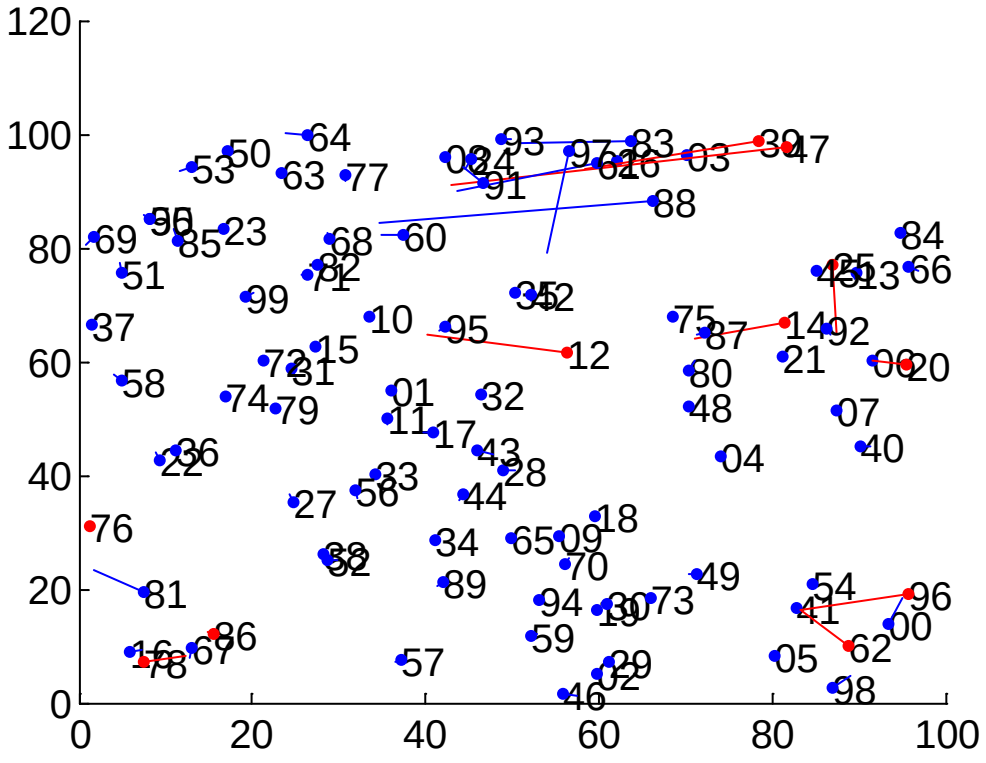
Function name:	QUINT_ERROR
Function call:	[m45, m45s] = quint_error(C, S, D1, D2, D3, D4, D5, D6, dhq, dhqs, dqq)
Input:	C: 5 x 5 matrix with inter-node range observations S: 5 x 5 matrix with the corresponding variances D1…D6: barycentric coordinates of the two tetrahedrons dhq (dhqs): squared height difference when nodes 4 and 5 are on the same (other) side of the base plane 1,2,3 dqq: distance of the foot-points of 4 and 5 in the plane 1,2,3
Output:	m45: mean error of the difference between l45 calculated and r45 observed. m45s: same as m45, but this time point 5 being on the other side as point 4.
Description:	determines mean errors within a quint making use of redundant measurements.

3.7 Functions for Performance Assessment

3.7.1 Function ‘compare_results’

The function compare_results is used to compare two different point sets, which are stored in two different data files using the positioning format ‘P’. If one ID number does occur in both data sets, the coordinates of that node are compared, a deviation vector is determined. A statistic of the network deviations is calculated and the key statistical parameters are returned. The compare_results function also saves the vectors in a file and plots the results on a graphical display.

Function name:	COMPARE_RESULTS
Function call:	[avd, msd, maxd, mp, avdc, msdc, maxdc, mpc] = compare_results (old,new)
Input:	old: name of data file with true (i.e. generated node positions) new: name of data file with restored node positions
Output:	avd: average position deviation msd: mean position deviation maxd: maximum deviation mp: mean point error of adjustment (a priory) avdc: average position deviation of coarse positioning mode msdc: mean position deviation of coarse positioning mode maxdc: maximum deviation of coarse positioning mode mpc: mean point error of adjustment (a priory) of coarse positioning mode
Description:	reads the two data files specified. Compares the two point sets and plots the points. Creates a new file 'comparison.erg' and writes the results (e.g. positional deviations) in it.



Example plot of the deviation vectors, generated by the compare_results function. The blue nodes and vectors are result of precise geodetic positioning and the red nodes and vectors result from the coarse positioning mode.

3.7.2 Function 'test_orgdata'

The function test_orgdata stands for 'test_with_original_data' and is used for testing the software, i.e. to find errors in the code. Instead of using the determined approximate node positions, it uses the 'true' node positions as they were generated in the 'input_file'. An adjustment is carried out based on these node positions. If the code is error free, the result (mean point error, position deviations etc.) should be the same as of 'compare_results'.

Function name:	TEST_ORGDATA
Function call:	[avd, msd, maxd, mp, avdc, msdc, maxdc, mpc] = test_orgdata(old,new,R)
Input:	same as in compare_results, see 3.6.1, but additionally R: Range-Structure with ranging information, see 2.6.2.
Output:	same as in compare_results, see 3.6.1.
Description:	Gives the true quality indicators of the node positions, thereby avoiding the actual lateration and clusterisation algorithms.

4 Components Configuration Description

5 References

Horn, B (1987): Closed-from solution of absolute orientation using unit quaternions. Journal of Opt. Soc. Amer., vol. A-4, pp. 629–642, 1987.

Mautz, R., Ochieng, W.Y., Brodin, G., Kemp, A. (2010): "3D Wireless Network Localization from Inconsistent Distance Observations", accepted for publication in Ad Hoc & Sensor Wireless Networks.

Thomas, F and Ros, L (2005): Revisiting Trilateration for Robot Localization, IEEE Transactions on Robotics, Vol. 21, No. 1, pp. 93-101, February 2005.

6 Directory

6.1 Acronym List

A_v	Position Availability (not available, approximation available, available)
ID	Device ID
P	Position format
R	Range format
r	range [m]
Re	Reliability (no redundancy, some redundancy or reliable)
r_{max}	Maximal range [m]
RSSI	Received Signal Strength Indicator [dB]
S	Status of a node (fixed, static, moving, unknowns)
SIP	Session Initiation Protocol
av	Status of a range [available, not available but radio link]
t	Time stamp [s]
th	Threshold [m]
w	weight of observation
x	X-Coordinate [m]
y	Y-Coordinate [m]
z	Z-Coordinate [m]

α	Threshold confidence level [%]
σ	Point Variance [m]
σ_x	Variance in X [m]
σ_y	Variance in Y [m]
σ_z	Variance in Z [m]
PID	Point Identifiers; empty list with ID-number of all nodes