Quasi Deterministic Radio Channel Generator User Manual and Documentation

Quadriga
The Next Generation Radio Channel Model

Document Revision: v2.2.0
June 27, 2019

Fraunhofer Heinrich Hertz Institute
Wireless Communications and Networks
Einsteinufer 37, 10587 Berlin, Germany

e-mail: quadriga@hhi.fraunhofer.de
http://www.quadriga-channel-model.de

Fraunhofer
Heinrich Hertz Institute

Contributors

Editor:	Fraunhofer Heinrich Hertz Institute Wireless Communications and Networks Einsteinufer 37, 10587 Berlin, Germany
Contributing Authors:	Stephan Jaeckel, Leszek Raschkowski and Lars Thiele *Fraunhofer Heinrich Hertz Institute* Frank Burkhardt and Ernst Eberlein *Fraunhofer Institute for Integrated Circuits IIS*

Grants and Funding

This work was supported by

- the European Space Agency (ESA) in the Advanced Research in Telecommunications Systems (ARTES) programme under contract AO/1-5985/09/08/NL/LvH (Acronym: MIMOSA), [1]
  `http://artes.esa.int/projects/mimosa-characterisation-mimo-channel-mobile-satellite-systems`
- the German Federal Ministry of Economics and Technology (BMWi) in the national collaborative project IntelliSpektrum under contract 01ME11024
  `http://www.intellispektrum.de`
- the European Commission co-funded the project METIS as an Integrated Project under the Seventh Framework Programme for research and development (FP7)
  `http://www.metis2020.com`
- the GreenTouch consortium within the funded project "LSAS Channel Modelling"
  `http://www.greentouch.org`
- the European Commission co-funded the project mmMAGIC as an Integrated Project under the Horizon 2020 Programme for research and development (FP7)
  `https://5g-mmmagic.eu`
- the Celtic-Plus funded project REICOVAIR
  `https://www.celticplus.eu/project-reicovair`

Acknowledgements

The authors thank G. Sommerkorn, C. Schneider, M. Kaeske [Ilmenau University of Technology (IUT), Ilmenau, Germany] and V. Jungnickel [Heinrich Hertz Institute (HHI), Berlin, Germany] for the fruitful discussions on the QuaDRiGa channel model and the manuscript of this document.

How to Cite QuaDRiGa

[2] S. Jaeckel, L. Raschkowski, K. Börner, and L. Thiele, "QuaDRiGa: A 3-D multi-cell channel model with time evolution for enabling virtual field trials," *IEEE Trans. Antennas Propag.*, vol. 62, pp. 3242-3256, 2014.

[3] S. Jaeckel, L. Raschkowski, K. Börner, L. Thiele, F. Burkhardt and E. Eberlein, "QuaDRiGa - Quasi Deterministic Radio Channel Generator, User Manual and Documentation", Fraunhofer Heinrich Hertz Institute, Tech. Rep. v2.2.0, 2019.

Contents

List of Figures

List of Tables

List of Acronyms

1-D	one-dimensional
2-D	two-dimensional
3-D	three-dimensional
4-D	four-dimensional
3GPP	3rd generation partnership project
5G	fifth generation
ACF	autocorrelation function
AoA	azimuth angle of arrival
AoD	azimuth angle of departure
AS	angular spread
ASA	azimuth spread of arrival
ASD	azimuth spread of departure
ASE	average squared error
BP	break point
BS	base station
CDF	cumulative distribution function
CIR	channel impulse response
COST	European Cooperation in Science and Technology
D2D	device-to-device
DS	delay spread
EoA	elevation angle of arrival
EoD	elevation angle of departure
ESA	elevation spread of arrival
ESD	elevation spread of departure
FBS	first-bounce scatterer
GCS	global coordinate system
GR	ground reflection
GSCM	geometry-based stochastic channel model
JCF	joint correlation function
KF	Ricean K-factor
LBS	last-bounce scatterer
LHCP	left hand circular polarized
LOS	line of sight

LSF	large-scale fading
LSP	large-scale parameter
MIMO	multiple-input multiple-output
MIMOSA	MIMO over satellite
MPC	multipath component
MT	mobile terminal
NLOS	non-line of sight
NR	new radio
O2I	outdoor-to-indoor
OFDM	orthogonal frequency division multiplexing
P2P	peer-to-peer
PAS	power-angular spectrum
PDP	power delay profile
PG	path gain
PL	path loss
QuaDRiGa	quasi deterministic radio channel generator
RHCP	right hand circular polarized
RX	receiver
SCM	spatial channel model
SF	shadow fading
SISO	single input single output
SOS	sum-of-sinusoids
SSF	small-scale-fading
SSG	state sequence generator
STD	standard deviation
TX	transmitter
UMa	urban-macrocell
UMi	urban-microcell
UML	unified modeling language
WGS	world geodetic system
WINNER	Wireless World Initiative for New Radio
WSS	wide-sense stationary
WSSUS	wide sense stationary uncorrelated scattering
XPD	cross-polarization discrimination
XPR	cross polarization ratio
ZoA	zenith angle of arrival
ZoD	zenith angle of departure
ZSA	zenith angle spread of arrival
ZSD	zenith angle spread of departure

Glossary

base station (BS) 85

The term base station (BS) refers to a fixed transmitter which utilizes one or more transmit antennas to serve one or more MTs. BSs might further use *sectors* to increase the capacity. Usually, BSs operate independent of each other which might lead to inter-BS interference if they use the same time and frequency resource.

cluster 111, 114
A cluster describes an area where many scattering events occur simultaneously, e.g. at the foliage of trees or at a rough building wall. In the channel model, each scattering cluster is approximated by 20 single reflections. Each of those reflections has the same propagation delay.

drifting 102
Drifting occurs within a small area (about 20-30 m diameter) in which a specific "*cluster*" can be seen from the MT. Within this area the cluster position is fixed. Due to the mobility of the terminal the path length (resulting in a path delay) and the arrival angels change slowly, i.e. they "*drift*".

large-scale parameter (LSP) 91
The term "*large scale parameter*" refers to a set of specific properties of the propagation channel. Those are the "*delay spread*", the "*K-factor*", the "*shadow fading*", the "*cross-polarization ratio*", and four "*angular spread*"-values. Those properties can be extracted from channel sounding data. If a large amount of channel measurements is available for a specific propagation *scenario* and the LSPs can be calculated from those channels, statistics of the LSPs, e.g. their distribution and correlation properties can be obtained. A complete set of such statistical properties forms a "parameter table" that characterizes the *scenario*.

mobile terminal (MT) 85
Mobile terminals (MTs) are mobile receivers with one or more receive antennas. They are usually assigned to a serving BS which delivers data to the terminal.

multipath component (MPC) 85
Synonym for *path*.

path 113, 115
A path describes the way that a signal takes from the transmitter to the receiver. In the channel model, there is usually a direct, or LOS path, and several indirect, or NLOS paths. Indirect paths involve one or more scattering events which are described by clusters. However, paths do not describe single reflections but combine sub-paths that can not be separated in the delay domain. Usually, the channel model uses 6-25 paths to describe the propagation channel.

scatterer 102, 111
A scatterer describes a single reflection along a NLOS propagation path. Usually, several scatterers with a similar propagation delay and a narrow angular spread are combined into a "*(scattering) cluster*".

scattering cluster 85
Synonym for *cluster*.

scenario 92
In this thesis, the term *scenario* refers to a specific propagation environment such as "Urban macro-cell", "Urban satellite", "Indoor hotspot", etc. Usually, each propagation environment can be further split into LOS and NLOS propagation (e.g. "Urban macro-cell LOS" and "Urban macro-cell NLOS"), both of which might have very different properties. In the channel model, each *scenario* is fully specified by a parameter table.

segment . . . 86, 102
Segments are parts of a user trajectory in which the LSPs do not change considerably and where the channel keeps its WSS properties. Typical segment lengths are 5-30 m. It is assumed that within a segment, the scattering clusters are fixed.

sub-path . . . 101, 113
A sub-path is the exact way that a signal takes from the transmitter to the receiver. It contains at least one reflection. However, normally the channel model uses two scatterers (resulting in two reflections) to create a sub-path. 20 sub-paths are combined to a path. The LOS path has no sub-paths.

time evolution . . . 85, 102
Time evolution describes how the propagation channel changes (or evolves) with time. In the channel model, two effects are used to describe this time-dependency: *drifting* and the birth and death of scattering clusters during the transition between *segments*. The propagation environment is considered static and, thus, the model includes time-evolution only when the receiver is moving.

user . . . 92
Synonym for *mobile terminal*.

List of Symbols

$(.)^T$	Transpose of a matrix	110
γ	Polarization rotation angle for the linear NLOS polarization in [rad]	111, 112
λ	Wavelength in units of [m]	86, 104
ϕ	Azimuth angle in [rad]. ϕ can be used for ϕ^d or ϕ^a	75, 89, 106, 107
ϕ^a	Azimuth angle of arrival (AoA) in [rad]	101
ϕ^d	Azimuth angle of departure (AoD) in [rad]	101
$\hat{\phi}$	The offset angle between the path angle ϕ of the m^{th} sub-path in [degree]	101
ψ	Phase of a path in [rad]	104, 113
ρ	Correlation coefficient	88, 94
σ_τ	The RMS delay spread in units of [s]	92
τ	Delay of a MPC in units of [s]	103, 104
θ	Elevation angle in [rad]. θ can be used for θ^d or θ^a	75, 89, 106, 107
ϑ	Polarization rotation angle in [rad]	110
θ^a	Elevation angle of arrival (EoA) in [rad]	101
θ^d	Elevation angle of departure (EoD) in [rad]	101
$\mathbf{a}$	Vector pointing from the position of the LBS to the RX position	103
B	Bandwidth in units of [Hz]	86
c	Speed of Light	103, 104
$\mathbf{c}$	Representation of the departure or arrival angle in Cartesian coordinates	108
c_ϕ	The scenario-dependent cluster-wise RMS angular spread in [degree]	101
d	Length of a propagation path in [m]	103, 113
d_λ	Decorrelation distance in [m] where the autocorrelation falls below e^{-1}	88
$\mathbf{e}_{r,s}$	Vector from the RX position to RX antenna element r at snapshot s	103
$\mathbf{F}$	Polarimetric antenna response	107, 109, 112
f_S	Sampling Rate in [samples per meter]	87
f_T	Sampling Rate in [samples per second]	86
g	Channel coefficient in time domain	112
$\mathbf{J}$	Jones vector	107
k	Model parameter value, usually as a function of location or time	88

l	Path index, $l \in \{1, 2, ..., L\}$	102, 103
m	Sub-path index, $m \in \{1, 2, ..., M\}$	101, 102
$\mathcal{N}$	Normal distribution $\mathcal{N}(\mu, \sigma^2)$ with mean μ and STD σ	88, 111, 112
r	Receive antenna index; $r \in \{1, 2, ..., n_r\}$	102
$\mathbf{R}$	Rotation matrix	108, 109
$\mathbf{r}$	Vector pointing from the TX position to the RX position	103, 106
s	Snapshot index $s \in \{1, 2, ..., S\}$	102
t	Transmit antenna index; $t \in \{1, 2, ..., n_t\}$	102
$\mathcal{U}$	Continuous uniform distribution $\mathcal{U}(a, b)$ with minimum a and maximum b	107
v	Speed in [m/s]	86
Z	A random variable	107

References

[1] E. Eberlein, T. Heyn, F. Burkhardt, S. Jaeckel, L. Thiele, T. Haustein, G. Sommerkorn, M. Käske, C. Schneider, M. Dominguez, and J. Grotz, "Characterisation of the MIMO channel for mobile satellite systems (acronym: MIMOSA), TN8.2 – final report," Fraunhofer Institute for Integrated Circuits (IIS), Tech. Rep. v1.0, 2013.

[2] S. Jaeckel, L. Raschkowski, K. Börner, and L. Thiele, "QuaDRiGa: A 3-D multi-cell channel model with time evolution for enabling virtual field trials," *IEEE Trans. Antennas Propag.*, vol. 62, pp. 3242–3256, 2014.

[3] S. Jaeckel, L. Raschkowski, K. Börner, L. Thiele, F. Burkhardt, and E. Eberlein, "QuaDRiGa - Quasi Deterministic Radio Channel Generator, User Manual and Documentation," Fraunhofer Heinrich Hertz Institute, Tech. Rep. v1.4.1-551, 2016.

[4] P. Kyösti, J. Meinilä, L. Hentilä *et al.*, "IST-4-027756 WINNER II D1.1.2 v.1.1: WINNER II channel models," Tech. Rep., 2007. [Online]. Available: http://www.ist-winner.org

[5] ITU-R P.527-3, "Electrical characteristics of the surface of the earth," Tech. Rep., 1992.

[6] P. Heino, J. Meinilä, P. Kyösti *et al.*, "CELTIC / CP5-026 D5.3: WINNER+ final channel models," Tech. Rep., 2010. [Online]. Available: http://projects.celtic-initiative.org/winner+

[7] C. Schneider, M. Narandzic, M. Käske, G. Sommerkorn, and R. Thomä, "Large scale parameter for the WINNER II channel model at 2.53 GHz in urban macro cell," *Proc. IEEE VTC '10 Spring*, 2010.

[8] M. Narandzic, C. Schneider, M. Käske, S. Jaeckel, G. Sommerkorn, and R. Thomä, "Large-scale parameters of wideband MIMO channel in urban multi-cell scenario," *Proc. EUCAP '11*, 2011.

[9] 3GPP TR 36.873 v12.5.0, "Study on 3D channel model for LTE," Tech. Rep., 2017.

[10] 3GPP TR 38.901 v14.1.0, "Study on channel model for frequencies from 0.5 to 100 GHz," Tech. Rep., 2017.

[11] X. Cai and G. B. Giannakis, "A two-dimensional channel simulation model for shadowing processes," *IEEE Trans. Veh. Technol.*, vol. 52, no. 6, pp. 1558–1567, 2003.

[12] S. Jaeckel, L. Raschkowski, F. Burkhardt, and L. Thiele, "Efficient sum-of-sinusoids based spatial consistency for the 3gpp new-radio channel model," *Proc. IEEE Globecom Workshops '18*, 2018.

[13] S. Jaeckel, L. Raschkowski, S. Wu, L. Thiele, and W. Keusgen, "An explicit ground reflection model for mm-wave channels," *Proc. IEEE WCNC Workshops '17*, 2017. [Online]. Available: https://doi.org/10.1109/WCNCW.2017.7919093

[14] 3GPP TR 38.901 v15.0.0, “Study on channel model for frequencies from 0.5 to 100 GHz,” Tech. Rep., 2018.

[15] H2020-ICT-671650-mmMAGIC/D2.2, “mmMAGIC D2.2 - measurement results and final mmMAGIC channel models,” Tech. Rep., 2017.

[16] S. Jaeckel, N. Turay, L. Raschkowski, L. Thiele, R. V. andMarko Sonkki, V. Hovinen, F. Burkhardt, P. Karunakaran, and T. Heyn, “Industrial indoor measurements from 2-6 ghz for the 3gpp-nr and quadriga channel model,” *Proc. IEEE VTC’19 Fall (Submitted)*, 2019.

[17] L. Raschkowski, S. Jaeckel, F. Undi, L. Thiele, W. Keusgen, B. Pitakdumrongkija, and M. Ariyoshi, “Directional propagation measurements and modeling in an urban environment at 3.7 GHz,” *Proc. ACSSC ’16*, pp. 1799–1803, 2016.

[18] S. Jaeckel, “Quasi-deterministic channel modeling and experimental validation in cooperative and massive MIMO deployment topologies,” Ph.D. dissertation, TU Ilmenau, 2017. [Online]. Available: https://www.db-thueringen.de/receive/dbt_mods_00032895

[19] E. Eberlein, F. Burkhardt, G. Sommerkorn, S. Jaeckel, and R. Prieto-Cerdeira, “MIMOSA - analysis of the MIMO channel for LMS systems,” *Space Communications*, vol. 22, no. 2-4, pp. 145–158, 2013.

[20] F. Burkhardt, S. Jaeckel, E. Eberlein, and R. Prieto-Cerdeira, “QuaDRiGa: a MIMO channel model for land mobile satellite,” *8th European Conference on Antennas and Propagation (EuCAP)*, 2014.

[21] G. F. Masters and S. F. Gregson, “Coordinate system plotting for antenna measurements,” *AMTA Annual Meeting & Symposium*, 2007.

[22] 3GPP TR 25.996 v14.0.0, “Spatial channel model for multiple input multiple output (MIMO) simulations,” Tech. Rep., 2017.

[23] L. Correia, Ed., *Mobile Broadband Multimedia Networks.* Elsevier, 2006, ch. 6.8: The COST 273 MIMO channel model, pp. 364–383.

[24] 3GPP TR 36.873 v12.2.0, “Study on 3D channel model for LTE,” Tech. Rep., 2015.

[25] H. Xiao, A. Burr, and L. Song, “A time-variant wideband spatial channel model based on the 3gpp model,” *Proc. IEEE VCT ’06 Fall*, 2006.

[26] D. Baum, J. Hansen, and J. Salo, “An interim channel model for beyond-3G systems,” *Proc. IEEE VCT ’05 Spring*, vol. 5, pp. 3132–3136, 2005.

[27] M. Shafi, M. Zhang, A. Moustakas, P. Smith, A. Molisch, F. Tufvesson, and S. Simon, “Polarized MIMO channels in 3-D: models, measurements and mutual information,” *IEEE J. Sel. Areas Commun.*, vol. 24, pp. 514–527, Mar. 2006.

[28] C. Oestges, N. Czink, P. D. Doncker *et al.*, *Pervasive Mobile and Ambient Wireless Communications (COST Action 2100).* Springer, 2012, ch. 3: Radio Channel Modeling for 4G Networks, pp. 67–147.

[29] A. Zajic, G. Stuber, T. Pratt, and S. Nguyen, “Wideband MIMO mobile-to-mobile channels: Geometry-based statistical modeling with experimental verification,” *IEEE Trans. Veh. Technol.*, vol. 58, no. 2, pp. 517–534, 2009.

[30] M. R. Andrews, P. P. Mitra, and R. de Carvalho, “Tripling the capacity of wireless communications using electromagnetic polarization,” *Nature*, vol. 409, pp. 316–318, Jan 2001.

[31] S. Jaeckel, L. Thiele, and V. Jungnickel, “Interference limited MIMO measurements,” *Proc. IEEE VTC ’10 Spring*, 2010.

[32] M. Narandzic, M. Käske, C. Schneider, M. Milojevic, M. Landmann, G. Sommerkorn, and R. Thomä, "3D-antenna array model for IST-WINNER channel simulations," *Proc. IEEE VTC '07 Spring*, pp. 319–323, 2007.

[33] C. Oestges, B. Clerckx, M. Guillaud, and M. Debbah, "Dual-polarized wireless communications: From propagation models to system performance evaluation," *IEEE Trans. Wireless Commun.*, vol. 7, no. 10, pp. 4019–4031, 2008.

[34] Y. Zhou, S. Rondineau, D. Popovic, A. Sayeed, and Z. Popovic, "Virtual channel space-time processing with dual-polarization discrete lens antenna arrays," *IEEE Trans. Antennas Propag.*, vol. 53, pp. 2444–2455, Aug. 2005.

[35] F. Quitin, C. Oestges, F. Horlin, and P. De Doncker, "Multipolarized MIMO channel characteristics: Analytical study and experimental results," *IEEE Trans. Antennas Propag.*, vol. 57, pp. 2739–2745, 2009.

[36] R. C. Jones, "A new calculus for the treatment of optical systems, i. description and discussion of the calculus," *Journal of the Optical Society of America*, vol. 31, pp. 488–493, July 1941.

[37] J. Poutanen, K. Haneda, L. Liu, C. Oestges, F. Tufvesson, and P. Vainikainen, "Parameterization of the COST 2100 MIMO channel model in indoor scenarios," *Proc. EUCAP '11*, pp. 3606–3610, 2011.

[38] M. Zhu, F. Tufvesson, and G. Eriksson, "The COST 2100 channel model: Parameterization and validation based on outdoor MIMO measurements at 300 MHz," Lund University, Sweden, Tech. Rep., 2012.

[39] N. Czink, T. Zemen, J.-P. Nuutinen, J. Ylitalo, and E. Bonek, "A time-variant MIMO channel model directly parametrised from measurements," *EURASIP J. Wireless Commun. Netw.*, no. 2009:687238, 2009.

[40] K. Saito, K. Kitao, T. Imai, Y. Okano, and S. Miura, "The modeling method of time-correlated mimo channels using the particle filter," *Proc. IEEE VCT '11 Spring*, 2011.

[41] W. Wang, T. Jost, U. Fiebig, and W. Koch, "Time-variant channel modeling with application to mobile radio based positioning," *Proc. IEEE GLOBECOM '12*, pp. 5038–5043, 2012.

[42] [Online]. Available: http://www.quadriga-channel-model.de

[43] S. Gregson, J. McCormick, and C. Parini, *Principles of Planar Near-Field Antenna Measurements.* IET, 2007.

[44] M. Gudmundson, "Correlation model for shadow fading in mobile radio systems," *IET Electron Lett.*, vol. 27, no. 23, pp. 2145–2146, November 1991.

[45] M. Pätzold, N. Avazov, and V. D. Nguyen, "Design of measurement-based correlation models for shadow fading," pp. 112–117, Oct 2010.

[46] K. Bakowski and K. Wesolowski, "Change the channel," *IEEE Veh. Technol. Mag.*, vol. 6, pp. 82–91, 2011.

[47] T. Jamsa and P. Kyosti, "Device-to-device extension to geometry-based stochastic channel models," *Antennas and Propagation (EuCAP), 2015 9th European Conference on*, pp. 1–4, April 2015.

[48] M. Patzold, U. Killat, and F. Laue, "A deterministic digital simulation model for suzuki processes with application to a shadowed rayleigh land mobile radio channel," *IEEE Transactions on Vehicular Technology*, vol. 45, no. 2, pp. 318–331, May 1996.

[49] Z. Wang, E. Tameh, and A. Nix, “A sum-of-sinusoids based simulation model for the joint shadowing process in urban peer-to-peer radio channels,” *Proc. IEEE VTC ’05 Fall*, vol. 3, pp. 1732–1736, Sept 2005.

[50] M. Deserno, “How to generate equidistributed points on the surface of a sphere,” Max-Planck-Institut für Polymerforschung, Tech. Rep., 2004.

[51] P. Hoeher, “A statistical discrete-time model for the wssus multipath channel,” *IEEE Transactions on Vehicular Technology*, vol. 41, no. 4, pp. 461–468, 1992.

[52] A. Algans, K. Pedersen, and P. Mogensen, “Experimental analysis of the joint statistical properties of azimuth spread, delay spread, and shadow fading,” *IEEE J. Sel. Areas Commun.*, vol. 20, no. 3, pp. 523–531, 2002.

[53] L. Greenstein, V. Erceg, Y. Yeh, and M. Clark, “A new path-gain/delay-spread propagation model for digital cellular channels,” *IEEE Trans. Veh. Technol.*, vol. 46, no. 2, pp. 477–485, 1997.

[54] N. J. Higham, “Newton’s method for the matrix square root,” *Mathematics of Computation*, vol. 46, no. 174, pp. 537–549, 1986. [Online]. Available: http://www.jstor.org/stable/2007992

[55] 3GPP TR 25.996 v6.1.0, “Spatial channel model for multiple input multiple output (MIMO) simulations,” Tech. Rep., 2003.

[56] A. Ludwig, “The definition of cross-polarization,” *IEEE Trans. Antennas Propagation*, vol. AP-21, pp. 116–119, 1973.

[57] C. Oestges, V. Erceg, and A. Paulraj, “Propagation modeling of MIMO multipolarized fixed wireless channels,” *IEEE Trans. Veh. Technol.*, vol. 53, pp. 644–654, May 2004.

[58] V. Erceg, H. Sampath, and S. Catreux-Erceg, “Dual-polarization versus single-polarization MIMO channel measurement results and modeling,” *IEEE Trans. Wireless Commun.*, vol. 5, pp. 28–33, Jan. 2006.

[59] M. Landmann, K. Sivasondhivat, J. Takada, and R. Thomä, “Polarisation behaviour of discrete multipath and diffuse scattering in urban environments at 4.5 GHz,” *EURASIP J. Wireless Commun. Netw.*, vol. 2007, no. 1, pp. 60–71, 2007.

[60] T. Svantesson, “A physical MIMO radio channel model for multi-element multi-polarized antenna systems,” *Proc. IEEE VTC’ 01 Fall*, vol. 2, pp. 1083–1087, 2001.

[61] L. Materum, J. Takada, I. Ida, and Y. Oishi, “Mobile station spatio-temporal multipath clustering of an estimated wideband MIMO double-directional channel of a small urban 4.5 GHz macrocell,” *EURASIP J. Wireless Commun. Netw.*, no. 2009:804021, 2009.

[62] F. Quitin, C. Oestges, F. Horlin, and P. De Doncker, “A polarized clustered channel model for indoor multiantenna systems at 3.6 GHz,” *IEEE Trans. Veh. Technol.*, vol. 59, no. 8, pp. 3685–3693, 2010.

[63] M. Hata, “Empirical formula for propagation loss in land mobile radio services,” *IEEE Trans. Veh. Technol.*, vol. 29, no. 3, pp. 317–325, 1980.

[64] R. J. Weiler, M. Peter, W. Keusgen, A. Kortke, and M. Wisotzki, “Millimeter-wave channel sounding of outdoor ground reflections,” in *2015 IEEE Radio and Wireless Symposium*, Jan 2015, pp. 95–97.

[65] M. Peter, R. J. Weiler, T. Kuhne *et al.*, “Millimeter-wave small-cell backhaul measurements and considerations on street-level deployment,” in *2015 IEEE Globecom Workshops*, 2015.

[66] 3GPP TR 38.900 v14.2.0, “Channel model for frequency spectrum above 6 GHz,” Tech. Rep., 2016.

[67] A. Goldsmith, *Wireless Communications.* Cambridge University Press, 2003.

[68] K. Yu, Q. Li, and M. Ho, “Measurement investigation of tap and cluster angular spreads at 5.2 GHz,” *IEEE Transactions on Antennas and Propagation*, vol. 53, no. 7, pp. 2156–2160, July 2005.

[69] ITU-R P.2040-1, “Effects of building materials and structures on radiowave propagation above about 100 mhz,” Tech. Rep., 2015.

[70] 3GPP TDOC R1-143469, “Summary of 3D-channel model calibration results,” Nokia Networks, Nokia Corporation, Tech. Rep., 2014. [Online]. Available: http://www.3gpp.org/DynaReport/TDocExMtg--R1-78--30657.htm

1 Introduction and Overview

1.1 Installation and System Requirements

QuaDRiGa v2.2.0 supports MATLAB and Octave. The installation is straightforward and it does not require any changes to your system settings. If you would like to use QuaDRiGa, just extract the ZIP-File containing the model files and add the "quadriga_src"-folder from the extracted archive to you MATLAB/Octave-Path. In MATLAB, this can be done by opening MATLAB and selecting "File" - "Set Path ..." from the menu. Then you can use the "Add folder ..." button to add QuaDRiGa to your MATLAB-Path. For Octave (Linux), you need to create a file named ".octaverc" in your home directory with the following content:

```
addpath('/[path to QuaDRiGa]/quadriga_src')
more off
```

The "`more off`" command enables the support for real-time progress reports which is by default disabled in Octave. In Windows, this file is located at "C:\[path to Octave]\share\octave\site\m\startup\octaverc".

Table 1: QuaDRiGa System Requirements

Requirement	Value
Required MATLAB version	7.12 (R2011a)
Required Octave version	4.0
Required toolboxes	none
Memory (RAM) requirement	1 GB
Processing power	1 GHz Single Core
Storage	50 MB
Operating System	Linux, Windows, Mac OS

The following table provides some compatibility tests for different operating systems, architectures, MATLAB versions, and QuaDRiGa versions.

Table 2: System Compatibility Tests

Operating System	MATLAB / Octave	Architecture	QuaDRiGa Version	Test result
Ubuntu 16.04	R2013a (9.1)	64 bit	2.2.0	works
	R2016b (8.1)	64 bit	2.2.0	works
	Octave 4.0.0	64 bit	2.2.0	works
	Octave 5.1.0	64 bit	2.2.0	works
Windows 7	R2016a (9.0)	64 bit	2.0.0	works
	Octave 4.2.1	64 bit	1.9.0	works

1.2 General Remarks

This document gives a detailed overview of the `QuaDRiGa` channel model and its implementation details. The model has been evolved from the Wireless World Initiative for New Radio (WINNER) channel model described in WINNER II deliverable D1.1.2 v.1.1 [4]. This document covers only the model itself. Measurement campaigns covering the extraction of suitable parameters can be found in the WINNER documentation [4, 6] or other publications such as [7, 8]. Furthermore, the MIMOSA project [1] covers the model development and parameter extraction for land-mobile satellite channels.

Figure 1 gives an overview of a family of geometry-based stochastic channel models (GSCMs), starting with the 3rd generation partnership project (3GPP)-spatial channel model (SCM) in 2003. Work on QuaDRiGa

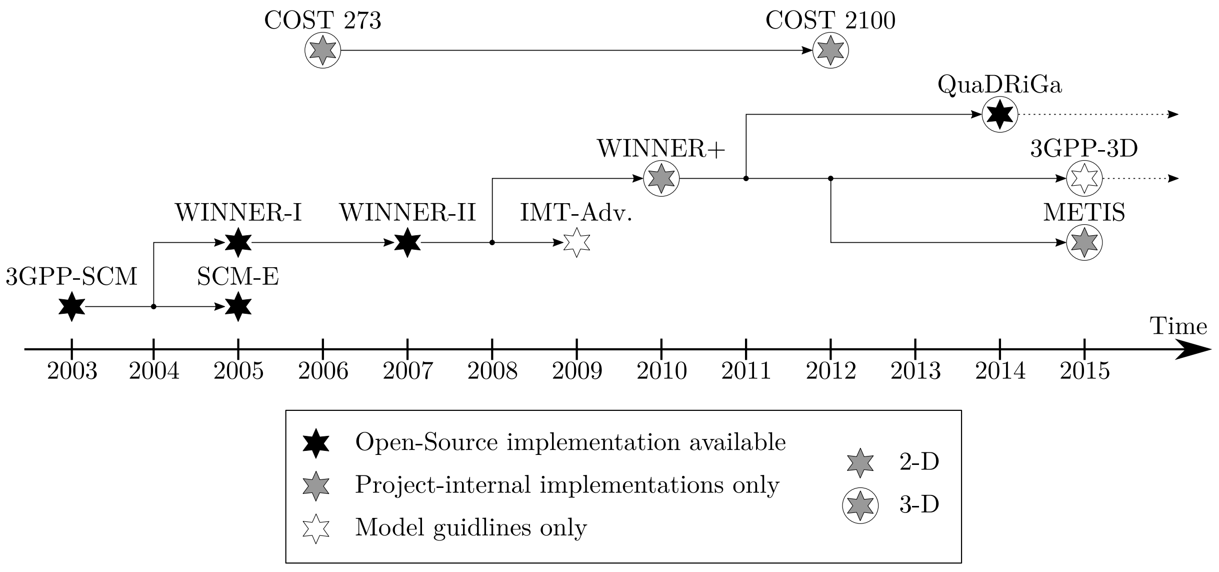


Figure 1: Evolution of GSCMs

started in 2011, after the end of the 3rd phase of the WINNER project. One year later, in 2012, 3GPP and the European-funded research project METIS started working on an evolution of the SCM, which later became commonly known as the 3GPP-3D channel model [9]. The latest 3GPP model [10] then extended the model towards mm-wave channels. However, the core components, e.g. the small-scale-fading (SSF) model, of this new model are in many parts identical to the WINNER+ model, which was also the baseline for the quasi deterministic radio channel generator (QuaDRiGa). Hence, QuaDRiGa can be regarded a 3GPP-3D and 3GPP 38.901 reference implementation. A mandatory part of the 3GPP-3D model is a calibration phase, were individual implementations of the 3GPP contributors have to create a set of metrics which show that the model implementation fulfills the requirements. This calibration exercise was also performed using QuaDRiGa. The results can be found in section 5.

The `QuaDRiGa` channel model follows a geometry-based stochastic channel modeling approach, which allows the creation of an arbitrary double directional radio channel. The channel model is antenna independent, i.e. different antenna configurations and different element patterns can be inserted. The channel parameters are determined stochastically, based on statistical distributions extracted from channel measurements. The distributions are defined for, e.g. delay spread, delay values, angle spread, shadow fading, and cross-polarization ratio. For each channel segment the channel parameters are calculated from the distributions. Specific channel realizations are generated by summing contributions of rays with specific channel parameters like delay, power, angle-of-arrival and angle-of-departure. Different scenarios are modeled by using the same approach, but different parameters. The basic features of the model approach can be summarized as follows:

- Support of freely configurable network layouts with multiple transmitters and receivers
- Scalability from a single input single output (SISO) or multiple-input multiple-output (MIMO) link to a multi-link MIMO scenario
- Same modeling approach indoor, outdoor, and satellite environments as well as combinations of them
- Support of a frequency range of 450 MHz to 100 GHz with up to 1 GHz RF bandwidth (additional frequency bands can be modeled as well, if suitable parameter tables are available)
- Support of multi-antenna technologies, polarization, multi-user, multi-cell, and multi-hop networks
- Smooth time evolution of large-scale and small-scale channel parameters including the transition between different scenarios
- High accuracy for the calculation of the polarization characteristics
- 3D model of antennas and propagation environment
- Support for massive MIMO antennas, both at the BS and mobile terminal (MT)

The `QuaDRiGa` channel model largely extends the WINNER+ and the 3GPP-3D model to support several new features that were originally not included. These are

- Time evolution
  Short term time evolution of the channel coefficients is realized by updating the delays, the departure- and arrival angles, the polarization, the shadow fading and the K-Factor based on the position of the terminal.
- Scenario transitions
  When the MT moves through the fading channel, it may pass through several different scenarios. `QuaDRiGa` supports smooth transitions between adjacent channel segments. This is used to emulate long term time evolution and allows the simulation of e.g. handover scenarios.
- Variable speeds for mobile terminals
  `QuaDRiGa` supports variable speeds including accelerating and slowing down of mobile terminals.
- Common framework for LOS and NLOS simulations
  In WINNER, line of sight (LOS) and non-line of sight (NLOS) scenarios were treated differently. `QuaDRiGa` used the same method for both scenarios types. This reduces the model complexity and enables freely configurable multicell scenarios. E.g. one MT can see two BSs, one in LOS and another in NLOS.
- Geometric polarization
  The polarizations for the LOS and for the NLOS case is now calculated based on a ray-geometric approach.
- Improved method for calculating correlated large-scale parameters (LSPs)
  The WINNER model calculates maps of correlated parameter values using filtered random fields. `QuaDRiGa` uses the sum-of-sinusoids method [11] to generate spatially correlated LSPs and spatially correlated SSF.
- New functions for modifying antenna patterns
  Antenna patterns can now be freely rotated in 3D-coordinates while maintaining the polarization properties. By default, individual antenna elements have individual antenna radiation patterns in azimuth and elevation direction. Those can also be imported from anechoic chamber measurements. The model further supports arbitrary array antenna structures where the elements can be placed in 3D coordinates. Hence, dual-polarized 2D or even 3D array structures both at the transmitter and receiver are supported.
- New MATLAB / Octave implementation
  The MATLAB code was completely rewritten. The implementations now fosters object oriented programming and object handles. This increases the performance significantly and lowers the memory usage.

1.3 Introduction to QuaDRiGa

QuaDRiGa (QUAsi Deterministic RadIo channel GenerAtor) was developed to enable the modeling of MIMO radio channels for specific network configurations, such as indoor, satellite or heterogeneous configurations.

Besides being a fully-fledged three dimensional geometry-based stochastic channel model, QuaDRiGa contains a collection of features created in SCM and WINNER channel models along with novel modeling approaches which provide features to enable quasi-deterministic multi-link tracking of users (receiver) movements in changing environments.

The main features of QuaDRiGa are:

- Three dimensional propagation (antenna modeling, geometric polarization, scattering clusters),
- Continuous time evolution,
- Spatially correlated large and small-scale-fading,
- Transitions between varying propagation scenarios

The QuaDRiGa approach can be understood as a "statistical ray-tracing model". Unlike the classical ray tracing approach, it does not use an exact geometric representation of the environment but distributes the positions of the scattering clusters (the sources of indirect signals such as buildings or trees) randomly. A simplified overview of the model is depicted in Figure 3. For each path, the model derives the angle of departure (the angle between the transmitter and the scattering cluster), the angle of arrival (the angle between the receiver and the scattering cluster) and the total path length which results in a delay τ of the signal. For the sake of simplicity, only two paths are shown in the figure.

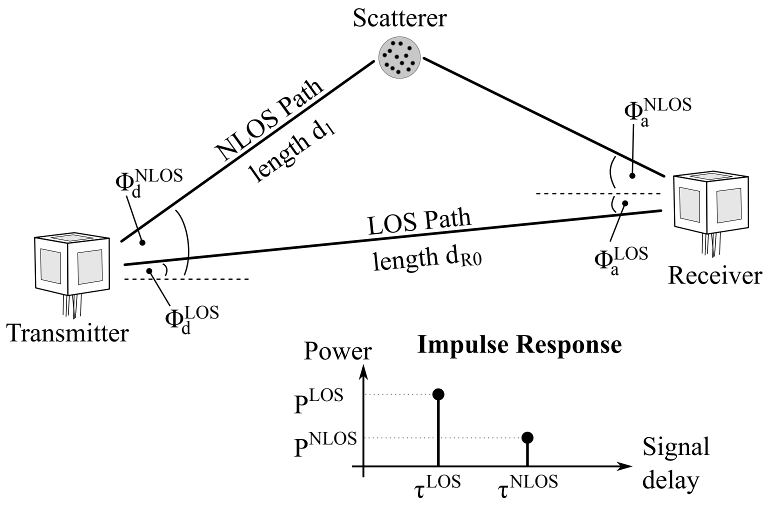


Figure 2: Simplified overview of the modeling approach used in `QuaDRiGa`

Terrestrial and Satellite scenarios can be modeled. For "Satellite to Earth" communication the angle of departure is identical for all clusters. The concept behind the model allows also the modeling of scenarios such as

- Earth to satellite
- Satellite systems with complementary ground components (CGC): Using several transmitters at different positions and simulating all propagation paths in one setup is supported.

The analysis of these scenarios was not in the scope of the MIMO over satellite (MIMOSA) project. This feature is not tested and especially no parameter sets are available yet.

In the following, the terms cluster, scattering cluster and scatterer are used synonymously. A cluster describes an area where many scattering events occur simultaneously, e.g. at the foliage of trees or at a rough building wall. In QuaDRiGa, each scattering cluster is approximated by 20 individual scatterers. Each one is modeled by a single reflection. The 20 signals can be resolved in spatial domain where they have a typical angular spread of 1-6°. However, they cannot be resolved in delay domain. Therefore, in the output of the channel model, these 20 signals (also named sub-paths) are combined into a single signal which is represented by a path. The difference to Rayleigh fading models, which use wide sense stationary uncorrelated scattering (WSSUS) taps instead of paths, is that each path has a very limited angular spread (1-6°) which also results in a narrow Doppler spectrum. The terms path, multipath component (MPC) and tap are also used synonymously in the QuaDRiGa documentation.

To emulate a rich scatting environment with a wider angular spread, many scattering clusters are created. QuaDRiGa has not upper limit for the amount of supported scattering clusters. However, depending on the angular spread and the amount of diffuse scatting (which is approximated by discrete clusters in QuaDRiGa), typical values are around 10 cluster for LOS propagation and 20 clusters for non-LOS. The positioning of the clusters is controlled by the environment angular spread and the delay spread. The environment angular spread has values of around 20-90° and is typically much larger than the per-cluster angular spread. However, even with many clusters, the Doppler spread is narrower in QuaDRiGa than when assuming pure Rayleigh fading. This is also in line with measurement results. It can be observed in the field that the main components arrive from selected angles and the classical Doppler spectrum's "Jakes" or Butterworth filter shaped characteristics are only valid as long term average and not valid for a short time interval.

To summarize:

- A typical propagation environment for channels at a carrier frequency below 6 GHz requires 8-20 clusters.
- Internally, each cluster is represented by 20 sub-paths, resulting in 160 - 400 sub-paths in total.
- Each sub-path is modeled as a single reflection.
- The 160 - 400 sub-paths are weighted by the antenna response. The 20 sub-paths for each cluster are summed up which results in 8-20 paths.
- For a MIMO system with multiple antennas at the transmitter and receiver, each path has as many channel coefficients, as there are antenna pairs. Hence, at the output, there are $n_{Path} \cdot n_{Rx} \cdot n_{Tx}$ channel coefficients.

1.4 Continuous time evolution

QuaDRiGa calculates the channel for each defined reception point. To generate a "time series" a continuous track of reception points can be defined. The arrival angles of the sub-paths play a crucial role for the time evolution because the phase changes are calculated deterministically based on the arrival angles. This results in a realistic Doppler spectrum.

The temporal evolution of the channel is modeled by two effects:

- drifting and
- birth and death of clusters.

Drifting (see Section 3.4) occurs within a small area (about 20-30 m diameter) in which a specific cluster can be seen from the MT. Within this area the cluster position is fixed. Due to the mobility of the terminal the path length (resulting in a path delay) and arrival angles change slowly.

Longer time-evolving channel sequences need to consider the birth and death of scattering clusters as well as transitions between different propagation environments. We address this by splitting the MT trajectory into segments. A segment can be seen as an interval in which the LSPs, e.g. the delay and angular spread, do not change considerably and where the channel keeps its wide-sense stationary (WSS) properties. Thus, the length of a segment depends on the decorrelation distances of the LSPs. We propose to limit the segment length to the average decorrelation distance. Typical values are around 20 m for LOS and 45 m for NLOS propagation. In the case where a state does not change over a long time, adjacent segment must have the same state. For example, a 200 m NLOS segment should be split into at least 4 NLOS sub-segments.

A set of clusters is generated independently for each segment. However, since the propagation channel does not change significantly from segment to segment, we need to include correlation. This is done by correlating the LSPs from segment to segment. For example, measurements show that the shadow fading (the average signal attenuation due to building, trees, etc.) is correlated over up to 100 m. Hence, we call all channel characteristics showing similarly slow changes LSPs.

With a segment length of 20 m, two neighboring segments of the same state will have similar receive-power. To get the correct correlation, QuaDRiGa correlates the shadow fading for a large area. This approach also contains cross-correlations to other LSPs such as the delay spread. For example, a shorter delay spread might result in a higher received power. Hence, there is a positive correlation between power and delays spread which is also included.

To get a continuous time-series of channel coefficients requires that the paths from different segments are combined at the output of the model. In between two segments clusters from the old segment disappear and new clusters appear. This is modeled by merging the channel coefficients of adjacent segments. The active time of a scattering cluster is confined within the combined length of two adjacent segments. The power of clusters from the old segment is ramped down and the power of new clusters is ramped up within the overlapping region of the two segments. The combination clusters to ramp up and down is modeled by a statistical process. Due to this approach, there are no sudden changes in the LSPs. For example, if the delay spread in the first segment is 400 ns and in the second it is 200 ns, then in the overlapping region, the delay spread (DS) slowly decreases till it reaches 200 ns. However, this requires a careful setup of the segments along the used trajectory. If the segments are too short, sudden changes cannot be excluded. This process is described in detail in Section 3.8.

1.5 QuaDRiGa program flow

For a propagation environment (e.g. urban, suburban, rural or tree-shadowing) typical channel characteristics are described by statistics of the LSPs. Those are the median and the standard deviation of the delay spread, angular spreads, shadow fading, Ricean K-Factor, as well as correlations between them. Additional parameters describe how fast certain properties of the channel change (i.e. the decorrelation distance). Those parameters are stored in configuration files which can be edited by the model user. Normally, the parameters are extracted from channel measurements. A detailed description of the model steps can be found Section 3.

1. The user of the model needs to configure the network layout. This includes:
   - Setting the transmitter position (e.g. the BS positions or the satellite orbital position)
   - Defining antenna properties for the transmitter and the receiver
   - Defining the user trajectory
   - Defining states (or segments) along the user trajectory
   - Assigning a propagation environment to each state

   Defining the user trajectory, states along the user trajectory and related parameters is performed by the state sequence generator (SSG). In the current implementation different SSGs are available:

   - Manual definition of all parameters by the user, e.g. definition of short tracks.
   - Statistical model for the "journey". A simple model (mainly designed for demonstration and testing purpose is included in the tutorial "satellite_channel")
   - Derive trajectory and state sequence from the measurement data.

2. Configuration files define the statistical properties of the LSPs. For each state (also called scenario) a set of properties is provided. Typically two configurations files are used.
   - One for the "good state" (also called LOS scenario)
   - The other for the "bad state" (NLOS scenario).

   For each state QuaDRiGa generates correlated "maps" for each LSP. For example, the delay spread in the file is defined as log-normal distributed with a range from 40 to 400 ns. QuaDRiGa translates this distribution in to a series of discrete values, e.g. 307 ns for segment 1, 152 ns for segment 2, 233 ns for segment 3 and so on. This is done for all LSPs.

3. The trajectory describes the position of the MT. For each segment of the trajectory, clusters are calculated according to the values of the LSPs at the MT position. The cluster positions are random within the limits given by the LSP. For example, a delay spread of 152 ns limits the distance between the clusters and the terminal.

4. Each cluster is split into 20 sub-paths and the arrival angles are calculated for each sub-path and for each positions of the terminal on the trajectory.

5. The antenna response for each of the arrival angles is calculated (the same holds for the departure angles). If there is more than one antenna at the transmitter- and/or receiver side, the calculation is repeated for each antenna.

6. The phases are calculated based on the position of the terminal antennas in relation to the clusters. The terminal trajectory defines how the phases change. This results in the Doppler spread.

7. The coefficients of the 20 sub-paths are summed (the output are paths). If there is more than one antenna and depending on the phase, this sum results in a different received power for each antenna-pair. At this point, the MIMO channel response is created.

8. The channel coefficients of adjacent segments are combined (merged). This includes the birth/death process of clusters. Additionally, different speeds of the terminal can be emulated by interpolation of the channel coefficients.

9. The channel coefficients together with the path delays are formatted and returned to the user for further analysis.

1.6 Description of modeling of different reception conditions by means of a typical drive course

This section describes some of the key features of the model using a real world example. A detailed introduction with a variety of tutorials, test cases and interface descriptions then follows in section 4. The later part of the document then focusses on the mathematical models behind the software and the assumptions made.

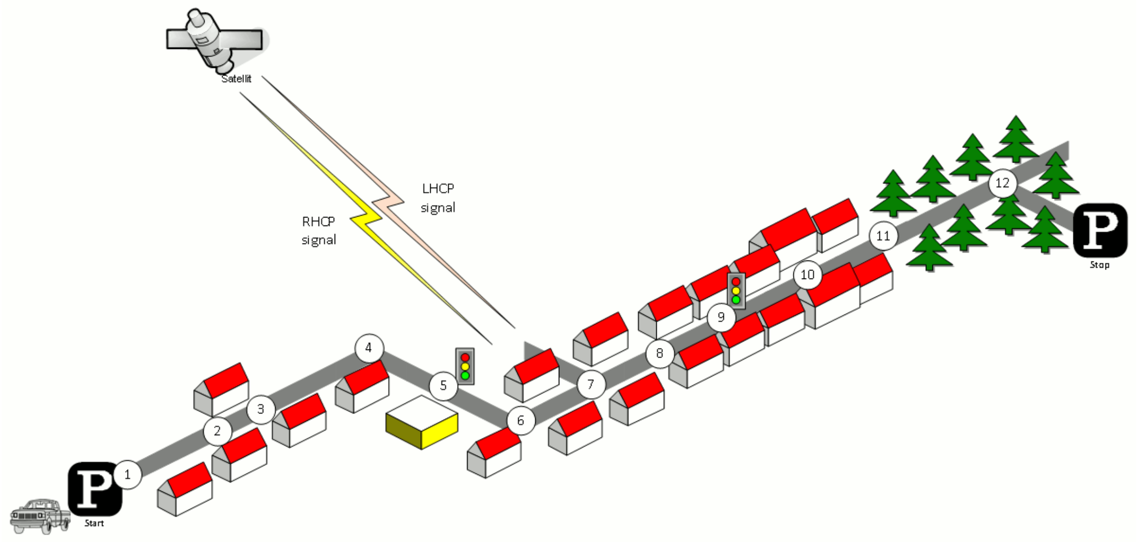


Figure 3: Typical driving course: From home to woodland parking site on the village outskirts

The different effects along the track can be summarized as follows:

1. Start environment: Urban, LOS reception of satellite signal
2. LOS → NLOS change
3. NLOS → LOS change
4. Turning off without change in reception condition (LOS)
5. Stopping at traffic light (LOS)
6. Turning off with change of reception condition (LOS → NLOS)
7. Crossing side street (NLOS → short LOS → NLOS)
8. Structural change in the environment without a change in the environment type (higher density of buildings but still the environment remains urban)
9. Stopping at traffic lights (NLOS)
10. Houses have the same characteristics as before but are further away from the street (urban environment with different reception characteristics)
11. Change of environment (Urban → Forest)
12. Turning off without change of environment (NLOS)

Each simulation run in QuaDRiGa is done in three (and an optional fourth) step.

1. Set up tracks, scenarios, antennas and network layout
2. Generate correlated LSPs
3. Calculate the channel coefficients
4. Post-processing (optional)

Those steps also need to be done for the above scenario. However, different aspects of the track are handled in different parts of the model. Additionally, the QuaDRiGa model supports two operating modes for handling the LSPs:

1. The first (default) mode generates the correlated LSPs automatically based on a scenario-specific parameter set. This is done in step 2 and involves so called parameter maps.

2. The manual mode does not generate LSPs automatically. Here, the user has to supply a list of parameters to the model. The step 2 thus to be implemented by the user.

Steps 1, 3 and 4 are identical for both modes. The following list describes the modeling of the observed effects along the track when using the automatic mode (1).

1. **Start environment: Urban, LOS reception of satellite signal**
   Each segment along the track gets assigned an environment. In the QuaDRiGa terminology, this is called a scenario. E.g. the first segment on the track is in the "Satellite-LOS-Urban"-scenario. The selection of the scenario is done during the first step (set up tracks, scenarios, antennas and network layout). After the model setup, the "automatic mode" generates a set of LSPs for this segment. I.e. the second step of the model calculates one value for each of the 7 LSPs. The third step then calculates a time-series of fading coefficients.

2. **LOS → NLOS change**
   A scenario change is defined along the track. E.g. the second segment along the track gets assigned the scenario "Satellite-NLOS-Urban". Now, a second set of LSPs is generated for all "Satellite-NLOS-Urban"-segments. We get a set of channel coefficients with different properties (e.g. more multipath components, lower K-Factor etc.). A smooth transition between the coefficients from the first segment and the second is realized by the ramping down the powers of the clusters of the old segment and ramping up the power of the new. This is implemented in step 4 (Post-processing).

3. **NLOS → LOS change**
   This is essentially the same as in point 2. However, since the third segment is also in the scenario

"Satellite-LOS-Urban", the parameters are extracted from the same spatially correlated generators as for the starting segment.

4. **Turning off without change in reception condition (LOS)**
QuaDRiGa supports free 3D-trajectories for the receiver. Thus, no new segment is needed - the terminal stays in the same segment as in point 3. However, we assume that the receive antenna is fixed to the terminal. Thus, if the car turns around, so does the antenna. Hence, the arrival angles of all clusters, including the direct path, change. This is modeled by a time-continuous update of the angles, delays and phases of each multipath component, also known as drifting. Due to the change of the arrival angles and the path-lengths, the terminal will also see a change in its Doppler-profile.

5. **Stopping at traffic light (LOS)**
QuaDRiGa performs all internal calculations at a constant speed. However, a stop of the car at a traffic light is realized by interpolating the channel coefficients in an additional post processing step (step 4). Here, the user needs to supply a movement profile that defines all acceleration, deceleration or stopping points along the track. An example is given in section **??**. Since the interpolation is an independent step, it makes no difference if the mobile terminal is in LOS or NLOS conditions.

6. **Turning off with change of reception condition (LOS $\rightarrow$ NLOS)**
This is realized by combining the methods of point 2 (scenario change) and point 4 (turning without change). The scenario change is directly in the curve. Thus, the LOS and the NLOS segments have an overlapping part where the cluster powers of the LOS segment ramp down and the NLOS clusters ramp up. The update of the angles, delays and phases is done for both segments in parallel.

7. **Crossing side street (NLOS $\rightarrow$ short LOS $\rightarrow$ NLOS)**
This is modeled by two successive scenario changes (NLOS-LOS and LOS-NLOS). For both changes, a new set of clusters is generated. However, since the parameters for the two NLOS-segments are extracted from the same map, they will be highly correlated. Thus, the two NLOS segments will have similar properties.

8. **Structural change in the environment without a change in the environment type** (higher density of buildings but still the environment remains urban)
This is not explicitly modeled. However, the "Satellite-NLOS-Urban" scenario covers a typical range of parameters. E.g. in a light NLOS area, the received power can be some dB higher compared to an area with denser buildings. The placement of light/dense areas on the map is random. Thus, different characteristics of the same scenario are modeled implicit. They are covered by the model, but the user has no influence on where specific characteristics occur on the map when using the automatic mode. An alternative would be to manually overwrite the automatically generated parameters or use the manual mode. In order to update the LSPs and use a new set of parameters, a new segment needs to be created. In this example, an environment change from "Satellite-NLOS-Urban" to the same "Satellite-NLOS-Urban" has to be created. Thus, a new set of LSPs is read from the map and new clusters are generated accordingly.

9. **Stopping at traffic lights (NLOS)**
This is the same as in point 5.

10. **Structural change in environment**
Houses have the same characteristics as before but are further away from the street (urban environment with different reception characteristics). This is the same as point 8.

11. **Change of environment (Urban $\rightarrow$ Forest)**
This is the same as in point 2. The segment on the track gets assigned the scenario "Satellite-Forest" and a third set of maps (15-21) is generated for the "Satellite-Forest"-segment. The parameters are drawn from the corresponding scenario distributions, new channel coefficients are calculated and the powers of the clusters are ramped up/down.

12. **Turning off without change of environment (NLOS)**
    Same as in point 4.

1.7 Compatibility with 3GPP models

This section provides an overview of the implemented 3GPP 36.873 [9] and 3GPP 38.901 [10] model components. Some modifications were made in order to make the models consistent. These modifications are described in the technical documentation (Section 3). 3GPP calibration results are reported in Section 5.

In addition to the small-scale-fading (SSF) model, which is in large parts identical to the WINNER+ model, 3GPP-3D specifies an antenna model, deployment scenarios, as well as path-loss models and parameter tables for urban-microcell (UMi) and urban-macrocell (UMa) deployments. All essential parts of the 3GPP-3D model have been implemented in QuaDRiGa as well. However, there are some differences between the two models which are explained in the following:

Differences between QuaDRiGa and the 3GPP-3D model

- **Coordinate system**
  The 3GPP-3D coordinate system is defined with respect to a spherical coordinate system where the zenith angle $\theta = 0^\circ$ points to the zenith and $\theta = 90^\circ$ points to the horizon. QuaDRiGa uses the geographic coordinate system where the elevation angle $\theta = 90^\circ$ points to the zenith and $\theta = 0^\circ$ points to the horizon. The conversion between the two is straight forward. To avoid confusion between the coordinate systems, 3GPP uses the term "zenith", i.e. zenith angle of arrival (ZoA), zenith angle of departure (ZoD), zenith angle spread of arrival (ZSA), zenith angle spread of departure (ZSD), while QuaDRiGa uses the term "elevation", i.e. elevation angle of arrival (EoA), elevation angle of departure (EoD), elevation spread of arrival (ESA), elevation spread of departure (ESD).

- **Delays, angles and path powers**
  3GPP-3D uses a heuristically determined Ricean K-factor dependent scaling constant in order to adjust the delays and angles in LOS scenarios (see [9], pp. 25). QuaDRiGa solves this differently by first assigning delays and path powers, including the Ricean K-factor (KF) power scaling. Then, the resulting DS is calculated and the path delays are scaled to the value from the large-scale fading (LSF) model. This avoids the heuristic scaling. See Section 3.3 for details.

- **Intra-cluster delay spread**
  The 3GPP-3D model splits the two strongest clusters into three sub-clusters (per cluster), with fixed delay offsets. However, when using the spatial consistency model from 3GPP 38.901 [10], the cluster-power changes as a function of the MT position. Hence, depending on where the MT is, the strongest clusters will be different. This will break the spatial consistency when for two neighboring positions, the two strongest clusters are different ones. Therefore, QuaDRiGa either splits ALL clusters into three sub-cluster or none. In both cases, spatial consistency is maintained.

- **Departure and arrival angles**
  3GPP-3D obtains the individual angles from mapping the path powers to a wrapped Gaussian or wrapped Laplacian power-angular spectrum (PAS). Then, heuristically determined scaling factors are used to adjust the angular values for a different number of paths and the Ricean K-factor (see [9], pp. 26, step 7 and 8). However, this approach breaks the input-output consistency of the angular spread, i.e. the angular spread calculated from the channel coefficients for an individual BS-MT link is not equal to the value given to the SSF model. Only the first-order statistics agree with each other. QuaDRiGa solves this by creating random angles, calculating the resulting angle spread, and scaling the angles to obtain the value from the LSF model (see Section 3.3). [1]

[1]The 3GPP-3D method is implemented as well and can be activated in the simulation setting of QuaDRiGa. However, there

- **Polarization model**
  QuaDRiGa has its own polarization model as described in Section 3.5. However, for the calibration, the model from 3GPP was used (see [9], pp. 26, step 11). The QuaDRiGa polarization model was originally introduced to correctly model ecliptic cross polarization ratios (XPRs) (e.g. for satellite channels), which is not covered well by the existing approach. The 3GPP / WINNER polarization model creates additional random phase shifts which effectively destroy ecliptic polarization in NLOS channels. These effects also change phase information in the channel coefficients - leading to a different singular-value spread in cross-polarized channels.

- **Wrapping method**
  QuaDRiGa does not implement a wrapping method for large deployments. Instead, MTs are only placed inside the inner ring of the BSs, which ensures that the interference from the outer rings is correctly modeled. An additional option is to add an additional ring of BS.[2] However, this roughly doubles the number of BSs in the deployment (111 instead of 57), adding additional memory requirements and computation time.

Baseline model features

The following table lists the implemented 3GPP baseline features:

Feature	3GPP Specification	QuaDRiGa v2.2.0
Coordinate system	TR 36.873 v12.5.0, Sec. 5.1, Page 7 TR 38.901 v14.1.0, Sec. 7.1, Page 14 Global coordinate system: Cartesian coordinates (in units of meters) with arbitrary origin. Local coordinate system: Spheric coordinates (elevation $\theta = 0°$ points to the zenith, $\theta = 90°$ points to the horizon)	Global coordinate system: Same as 3GPP. Local coordinate system: Geographic coordinates (elevation $\theta = 90°$ points to the zenith, $\theta = 0°$ points to the horizon)
Antenna modeling	TR 36.873 v12.5.0, Sec. 7.1, Page 17 TR 38.901 v14.1.0, Sec. 7.3, Page 21	3GPP antenna models are implemented in the 'qd_arrayant' class of the channel model. The 36.873 model is named `3gpp-3d` and the 38.901 model is named `3gpp-mmw`. See 'qd_arrayant.generate' for available antenna models.
Polarized antenna modeling	TR 36.873 v12.5.0, Sec. 7.1.1, Page 19 TR 38.901 v14.1.0, Sec. 7.3.2, Page 23 3GPP defines two model variants	The antenna polarization model in QuaDRiGa is equivalent to 'Model-1' in 3GPP (see Section 3.5.2). 3GPP 'Model-2' is not implemented.
Pathloss models	TR 36.873 v12.5.0, Sec. 7.2.1, Page 20 TR 38.901 v14.1.0, Sec. 7.4.1, Page 24	Path-loss models are implemented in 'qd_builder.get_pl'. The scenario-specific parameters are specified in the configuration files in the folder '`quadriga_src\config`'.
LOS probability	TR 36.873 v12.5.0, Sec. 7.2.2, Page 23 TR 38.901 v14.1.0, Sec. 7.4.2, Page 27	Line-Of-Sight (LOS) probability models are implemented in 'qd_layout.set_scenario'.
O2I penetration loss	TR 36.873 v12.5.0, Sec. 7.2.3, Page 24 TR 38.901 v14.1.0, Sec. 7.4.3, Page 27	TR 36.873 models included in path-loss formulas. TR 38.901 models implemented in 'qd_layout.gen_o2i_loss'
O2I car penetration loss	TR 38.901 v14.1.0, Sec. 7.4.3.2, Page 29	Not implemented
Autocorrelation of shadow fading	TR 36.873 v12.5.0, Sec. 7.2.4, Page 24 TR 38.901 v14.1.0, Sec. 7.4.4, Page 29	Implemented by the sum-of-sinusoids method in the 'qd_sos' class. See also Section 3.1.1 and [12] for details.

is no difference in the calibration results.

[2] There is an automatic function `qd_layout.generate('regular',37)` for this.

Feature	3GPP Specification	QuaDRiGa v2.2.0
Fast fading model	TR 36.873 v12.5.0, Sec. 7.3, Page 24 TR 38.901 v14.1.0, Sec. 7.5, Page 29 Baseline fast fading model. An alternative model is used for spatial consistency and multi-frequency simulations in TR 38.901.	Baseline model is not used by default! Enable by: qd_simulation_parameters.use_3GPP_baseline = 1 Some modifications have been made: 1. Matrix square-root is used instead of Cholesky decomposition for inter-parameter LSF correlation 2. Cluster delays, angles and powers are calculated as described in Section 3.3 3. Intra-cluster DS is applied to all clusters
NLOS polarization model	TR 36.873 v12.5.0, Page 35, Step 10 TR 38.901 v14.1.0, Page 35, Step 10	See Section 3.5. 3GPP polarization model is not used by default, but can be enabled by: qd_simulation_parameters.use_geometric_polarization = 0

Additional 3GPP 38.901 modeling components

In addition to the baseline features, 3GPP 38.901 specifies so-called additional features. The implementation status is as follows:

Feature	3GPP Specification	QuaDRiGa v2.2.0
Oxygen absorption	TR 38.901 v14.1.0, Sec. 7.6.1, Page 43	Not implemented
Large bandwidth and large antenna array	TR 38.901 v14.1.0, Sec. 7.6.2, Page 43	Large array antenna functionality is implemented and used by default. Large bandwidth extension is not implemented.
Spatial consistency procedure	TR 38.901 v14.1.0, Sec. 7.6.3.1, Page 45	Implemented by the sum-of-sinusoids method in the 'qd_sos' class. Requires alternative fast fading model (which is based on UT mobility modeling procedure B and multi-frequency simulations).
Spatially-consistent UT mobility modeling	TR 38.901 v14.1.0, Sec. 7.6.3.2, Page 46 3GPP defines two precedures (A and B)	QuaDRiGa uses drifting (see Section 3.4) for UT mobility modeling. This is similar to 3GPP procedure A, but not the same. 3GPP procedure B is implemented as well. See 'spatial_consistency' tutorial for details.
Spatially-consistent LOS/NLOS/indoor states and O2I parameters	TR 38.901 v14.1.0, Sec. 7.6.3.3, Page 50	Not implemented
Blockage	TR 38.901 v14.1.0, Sec. 7.6.3, Page 52	Not implemented
Multi-frequency simulations	TR 38.901 v14.1.0, Sec. 7.6.5, Page 57	Alternative channel generation method is implemented and enabled by default. See tutorial 'multi_frequency_simulations' for details.
Time-varying Doppler shift	TR 38.901 v14.1.0, Sec. 7.6.6, Page 60	QuaDRiGa uses drifting (see Section 3.4) which includes time-varying Doppler shifts. Variable MT speeds can be achieved as well. See tutorials 'time_evolution' and 'speed_profile_interpolation' for details.
UT rotation	TR 38.901 v14.1.0, Sec. 7.6.7, Page 60	Implemented. Can be controlled by the property 'orientation' of the 'qd_track' class.
Explicit ground reflection model	TR 38.901 v14.1.0, Sec. 7.6.8, Page 60	Implemented with minor modification. See [13] and tutorial 'ground_reflection' for details..

2 Software Structure

2.1 Overview

QuaDRiGa is implemented in MATLAB / Octave using an object oriented framework. The user interface is built upon classes which can be manipulated by the user. Each class contains fields to store data and methods to manipulate the data.

It is important to keep in mind that all classes in QuaDRiGa are “handle”-classes. This significantly reduces memory usage and speeds up the calculations. However, **all MATLAB variable names assigned to QuaDRiGa objects are pointers**. If you copy a variable (i.e. by assigning “**b = a**”), only the pointer is copied. “**a**” and “**b**” point to the same object in memory. If you change the values of “**b**”, the value of “**a**” is changed as well. This is somewhat different to the typical MATLAB behavior and might cause errors if not considered properly. Copying a QuaDRiGa object can be done by “**b = a.copy**”.

- **User input**
  The user inputs (Point 1 in the programm flow) are provided through the classes: “qd_simulation_parameters”, “qd_arrayant”, “qd_track”, and “qd_layout”.

  “**qd_simulation_parameters**” defines the general settings such as the center frequency and the sample density. It also enables and disables certain features of the model such as geometric polarization, spherical waves, and progress bars.

  “**qd_arrayant**” combines all functions needed to describe array antennas.

  “**qd_track**” is used to define user trajectories, states and segments.

  “**qd_layout**” combines the tracks and antenna properties together with further parameters such as satellite positions (e.g. for simulations including satellites).

- **Internal processing**
  Internal processing steps are done by the classes “qd_sos” and “qd_builder”.

  “**qd_sos**” is responsible for generating spatially correlated random variables based on the sum-of-sinusoids method.

  “**qd_builder**” creates the channel coefficients. It is responsible for generating LSPs for the cluster generation, the cluster generation, and the MIMO channels. It implements steps 2-7 of the program flow.

- **Model output**
  The final two steps (8 and 9) of the program flow are implemented in the class “**qd_channel**”. Objects of this class hold the data for the channel coefficients. The class also implements the channel merger, which creates long time evolving sequences out of the snapshots produced by the channel builder. Additional functions such as the transformation into frequency domain can help the user to further process the data.

An overview of the model software is depicted in Fig. 4. The unified modeling language (UML) class diagram of the `QuaDRiGa` channel model gives an overview of all the classes, methods and properties of the model. The class diagram serves as a reference for the following descriptions which also lists the methods that implement a specific functionality.

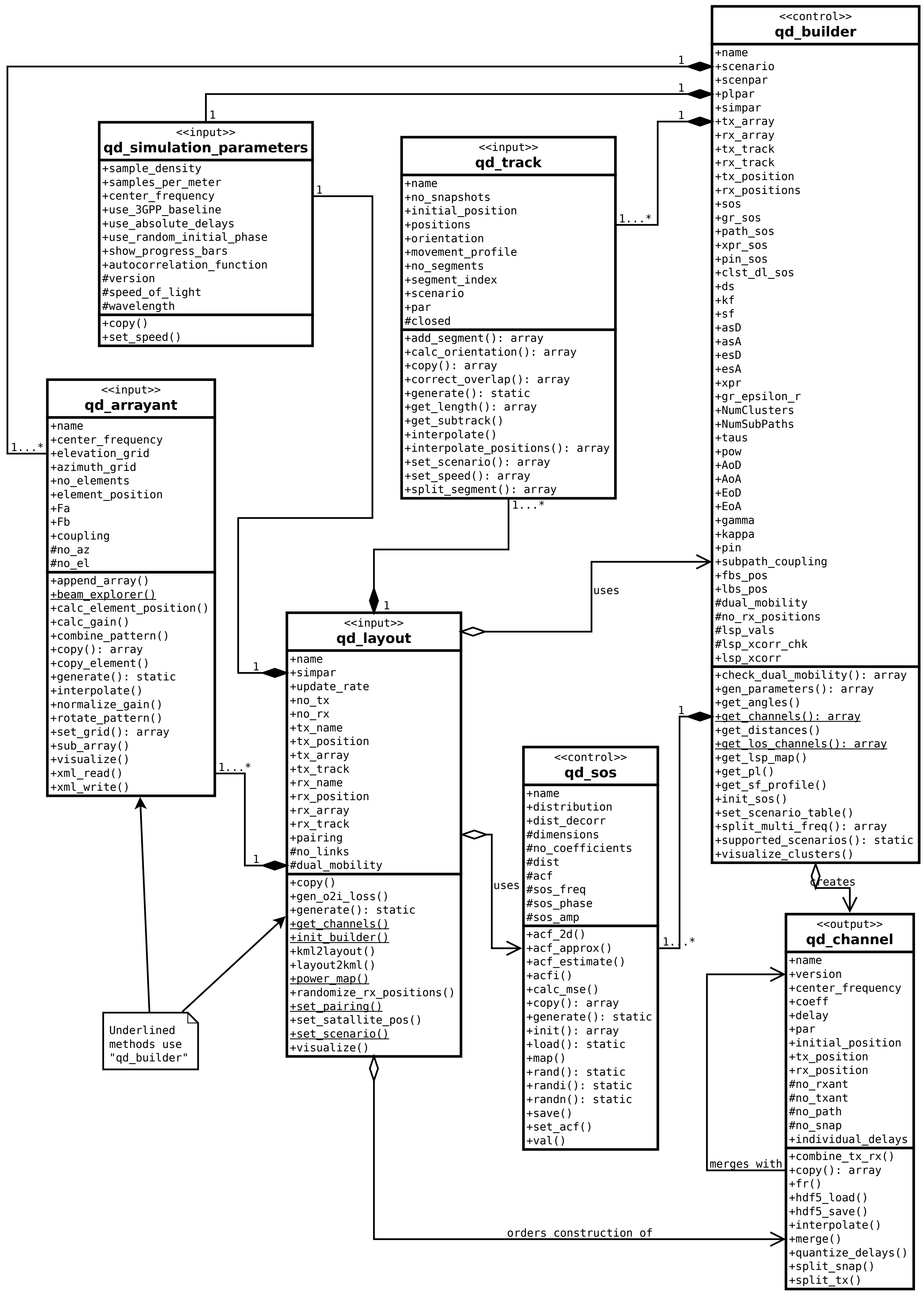


Figure 4: UML class diagram of the model software.

2.2 Description of Classes, Properties, and Methods

In the following, all properties and methods of the QuaDRiGa classes are described. For the methods, input and output variables are defined and explained. There are three types of methods: Standard methods require an instance of a class. They are printed in black without the class name:

par = **generate_parameters** (overlap, usage, check_parfiles, verbose)

Calling object	Single object
Description	Method description

Static methods can be called directly from the command line without creating an instance of the class first. They are printed in blue:

[h_array, mse, mse_pat] = **qd_array.import_pattern** (fVi, fHi)

Calling object	None (static method)
Description	Method description

The constructor is a special method that is called when the class name is used as a function, e.g. when calling `a = qd_array('dipole')`. There is only one constructor for each class. They are printed in blue.

h_array = **qd_array** (array_type, phi_3dB, theta_3dB, rear_gain)

Calling object	None (constructor)
Description	Method description

In addition, methods can either be called on single objects of a class or arrays of objects of the same class. This is specified by the field "Calling object". It can have three options:

- None (for constructors and static methods)
- Scalar object (method can only be called on single objects of a class)
- Object array (method can be called on single objects and arrays of objects of the same class)

Note that calls to object arrays work different in MATLAB and Octave. The following code works in MATLAB only:

```
1  a     = qd_arrayant;       % Create arrayant object
2  a(2)  = qd_arrayant;       % Create another arrayant object ("a" now has two elements)
3  a(2).name  = 'test';       % Assign name to second object (fails in Octave)
4  b     = a.copy;            % Copy the entire object array (fails in Octave)
```

Octave 4.0 and 4.2 have limited support for classes and contain some bugs. Line 3 fails because linear indexing of object arrays is buggy in those versions. Line 4 fails because the calling method is not implemented. The following code works on both platforms, MATLAB and Octave:

```
1  a     = qd_arrayant;       % Create arrayant object
2  a(1,2)  = qd_arrayant;     % Always use correct array indexing in Octave
3  a(1,2).name  = 'test';     % Assign name (works in Octave due to array indexing)
4  b     = copy(a);           % Call the copy method and provide "a" as input
```

2.2.1 Class “qd_simulation_parameters”

This class controls the simulation options and calculates constants for other classes.

Properties

sample_density	The number of samples per half-wave length Sampling density describes the number of samples per half-wave length. To fulfill the sampling theorem, the minimum sample density must be 2. For smaller values, interpolation of the channel for variable speed is not possible. On the other hand, high values significantly increase the computing time significantly. A good value is around 1.2 for single-mobility and 2.5 for dual-mobility.
samples_per_meter	Samples per meter This parameter is linked to the sample density by $$f_S = 2 \cdot f_C \cdot \frac{\text{SD}}{c}$$ where f_C is the carrier frequency in Hz, SD is the sample density and c is the speed of light.
center_frequency	Center frequency in [Hz]. For multi-frequency simulations, a vector of frequency values may be defined.
use_3GPP_baseline	Enables or disables the 3GPP baseline model. **use_3GPP_baseline = 0** (default) Disables the 3GPP baseline model and uses enhanced QuaDRiGa features: • This option uses spherical waves at both ends, the transmitter and the receiver. This method uses a multi-bounce model where the departure and arrival angles are matched such that the angular spreads stay consistent. • Uses the polarization rotation with an additional phase offset between the H and V component of the NLOS paths. The offset angle is calculated to match the XPR for circular polarization. **use_3GPP_baseline = 1** Disables all QuaDRiGa features that are not specified by 3GPP. • Applies rotating phasors to each path which emulates time varying Doppler characteristics. Paths are not tracked and mobility is limited to maximum 10 m. • The large-scale parameters (departure and arrival angles, shadow fading, delays, etc.) are not updated for terminal mobility. • The phases at the array antennas are calculated by a planar wave approximation. • Spatial consistency is not available. • Multi-frequency simulations are not supported. • No polarization rotation is calculated. The polarization transfer matrix contains random phasors scaled to match the XPR.
use_absolute_delays	Returns absolute delays in channel impulse response (CIR). By default, delays are calculated such that the LOS delay is normalized to 0. By setting ’`use_absolute_delays`’ to 1 or ’`true`’, the absolute path delays are included in ’`qd_channel.delays`’ at the output of the model.
use_random_ initial_phase	Initializes each path with a random initial phase By default, each path is initialized with a random phase (except the LOS path and the optional ground reflection). Setting ”`use_random_initial_phase`” to `false` disables this function. In this case, each path gets initialized with a zero-phase.
show_progress_bars	Show a progress bar on the MATLAB / Octave prompt. If this doesn’t work correctly, you need to enable real-time output by calling ”more off”.
autocorrelation_ function	The autocorrelation function for generating correlated model parameters. An autocorrelation function (ACF) is a description of the correlation vs. distance. This function is approximated by a Fourier series. The coefficients of the series can be used to generate spatially correlated random variables in the `qd_sos` class. There are 3 ACF types that can be selected. The coefficients are precomputed for 150, 300, 500, and 1000 sinusoids. • Exponential ACF (`Exp150`, `Exp300`, `Exp500`, `Exp1000`) • Gaussian ACF (`Gauss150`, `Gauss300`, `Gauss500`, `Gauss1000`) • Combined Gaussian and Exponential ACF (`Comb150`, `Comb300`, `Comb500`, `Comb1000`)
version speed_of_light wavelength	Version number of the current QuaDRiGa release (constant) Speed of light (constant) Carrier wavelength in [m] (read only)

Methods

h_simpar = **qd_simulation_parameters**		
Calling object	None (constructor)	
Description	Creates a new '`qd_simulation_parameters`' object with default settings	

out = **copy**		
Calling object	Object array	
Description	Creates a copy of the handle class object or array of objects. While the standard copy command creates new physical objects for each element of the object array (in case obj is an array of object handles), copy checks whether there are object handles pointing to the same object and keeps this information.	
Output	out	Copy of the current object or object array

set_speed (speed_kmh, sampling_rate_s)		
Calling object	Single object	
Description	This method can be used to automatically calculate the sample density for a given mobile speed	
Input	speed_kmh sampling_rate_s	speed in [km/h] channel update rate in [s]

2.2.2 Class "qd_arrayant"

This class combines all functions to create and edit array antennas. An array antenna is a set of single antenna elements, each having a specific beam pattern, that can be combined in any geometric arrangement. A set of synthetic arrays that allow simulations without providing your own antenna patterns is provided (see generate method for more details).

Properties

name	Name of the array antenna
center_frequency	Center frequency in [Hz]
no_elements	Number of antenna elements in the array Increasing the number of elements creates new elements which are initialized as copies of the first element. Decreasing the number of elements deletes the last elements from the array.
elevation_grid	Elevation angles (phi) in [rad] where samples of the field patterns are provided The field patterns are given in spherical coordinates. This variable provides the elevation sampling angles in radians ranging from $-\frac{\pi}{2}$ (downwards) to $\frac{\pi}{2}$ (upwards).
azimuth_grid	Azimuth angles (theta) in [rad] were samples of the field patterns are provided The field patterns are given in spherical coordinates. This variable provides the azimuth sampling angles in radians ranging from $-\pi$ to π.
element_position	Position of the antenna elements in local cartesian coordinates (using units of [m])
Fa	The first component of the antenna pattern contains the vertical component of the electric field given in spherical coordinates (aligned with the phi direction of the coordinate system). This variable is a tensor with dimensions [elevation, azimuth, element-index] describing the e-theta component of the far field of each antenna element in the array.
Fb	The second component of the antenna pattern contains the horizontal component of the electric field given in spherical coordinates (aligned with the theta direction of the coordinate system). This variable is a tensor with dimensions [elevation, azimuth, element-index] describing the e-phi component of the far field of each antenna element in the array.

coupling	Coupling matrix between elements This matrix describes a pre or postprocessing of the signals that are fed to the antenna elements. For example, in order to transmit a left hand circular polarized (LHCP) signal, two antenna elements are needed. They are then coupled by a matrix $$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ j \end{pmatrix}$$ The rows in the matrix correspond to the antenna elements, the columns to the signal ports. In this example, the antenna has one port, i.e. it is fed with one input signal. This signal is then split into two and fed to the two antenna elements where the second element radiates the signal with 90° phase shift. In a similiar fasion, it is possible to create fixed beamforming antennas and include crosstalk between antenna elements. By default, '`coupling`' is set to an identity matrix which indicates perfect isolation between the antenna elements.
no_az	Number of azimuth values
no_el	Number of elevation values

Methods

h_array = qd_arrayant (array_type, varargin)

Calling object	None (constructor)	
Description	Creates a new array object. The constructor calls `qd_arrayant.generate` to create new array antennas. If no input is specified, a vertically polarized omni-antenna is generated. See 'qd_arrayant.generate' for a description of the input parameters and the list of supported antenna types.	

append_array (a)

Calling object	Single object	
Description	Appends an array antenna to the existing one This method appends the array antenna given in "a" to the existing array object. The antenna patterns from "a" are copied to the calling object. For example, if the calling object has 3 elements and "a" has 2 elements, the two elements are added to the calling object which now has 5 elements. Element positions and coupling factors are copied as well.	
Input	a	The array object which is appended to the current array object (scalar object).

beam_explorer (Jp)

Calling object	Single object	
Description	Creates an interactive plot of the beam-forming capabilities of an array antenna When applying maximum-ratio transmission (MRT) to calculate the coupling weights of an array antenna, it is possible to direct a beam towards a given direction. However, the antenna geometry (i.e. the positions and the orientations of the individual elements) and the shape of the individual element-patterns will determine the overall shape of the beam and the existence and magnitude of so-called sidelobes. This method creates an interactive plot that uses the mouse pointer position to determine the target direction. The y-axis corresponds to the elevation direction and the x-axis corresponds to the azimuth direction relative to the local antenna coordinate system. Then, the method calculates the MRT-weights that direct the beam towards this position and applies it to the antenna pattern. The plot is then updated in real-time to visualize the radiated power using MRT beamforming. The maximum is normalized to 1, the minimum is normalized to 0. Ideally, the array antenna creates a single narrow beam that coincides exactly with the target direction, i.e. the maximum is always under the mouse pointer. However, design limitations (i.e., using planar or circular arrays, number of elements, etc.) will either lead to unwanted side-lobes or a widening of the main lobe. By clicking the left mouse button, the animation is paused and it is possible to use the data-pointer to read the values from the plot. Note: It is important to set the correct center frequency in the array object.	
Input	Jp	The polarization (Jones-vector) of the probe antenna, Default [1 ; 0]

[gain_dBi, pow_max] = **calc_gain** (i_element)

Calling object	Single object	
Description	Calculates the gain in dBi of the array antenna	
Input	i_element	A list of element indices.
Output	gain_dBi pow_max	Normalized Gain of the antenna in dBi. Maximum power in main beam direction in dBi.

element_position = **calc_element_position** (verbose)

Calling object	Single object	
Description	Calculates the element positions from the antenna patterns When an antenna pattern is measured in an anechoic chamber, the phase center, i.e. the point from which the electromagnetic radiation spreads spherically outward with the phase of the signal being equal at any point on the sphere, is relative to the rotation center of the positioner. This method calculates the element positions relative to the phase center of the array and adjusts the phases of the individual elements such that their phase center is centered at the element position. The maximum radius of the array antenna cannot exceed 20 wavelengths and the resolution of the position estimation is set to 0.02 wavelengths.	
Input	verbose	A value of 1 (default) shows a progress bar for the calculations. A value of 2 shows a plot visualizing the estimation process.
Output	element_position	Position of the antenna elements in local cartesian coordinates (using units of [m])

combine_pattern (center_frequency)

Calling object	Single object	
Description	Calculates a virtual pattern of the given array When the inputs of an array antenna are coupled (i.e. fed with the same signal), then it is possible to combine the elements of the array. This function calculates the virtual pattern by using the QuaDRiGa simulator. Individual coupling weights can be set in the "coupling" property of the `qd_arrayant` object. Phase offsets of the individual antenna elements due to their positions in the array ("element_position" property of the calling `qd_arrayant` object) are calculated for the phase center of the array.	
Input	center_frequency	The center frequency in [Hz]. If this input variable is not given, it is assumed that the element spacings in the "element_position" property of the calling arrayant object are given in multiples of the carrier wavelength.

out = **copy**

Calling object	Object array	
Description	Creates a copy of the handle class object or array of objects While the standard copy command creates new physical objects for each element of the object array (in case obj is an array of object handles), copy checks whether there are object handles pointing to the same object and keeps this information.	
Output	out	Copy of the current object or object array

copy_element (i_source, i_target)

Calling object	Single object	
Description	Creates a copy of an antenna element	
Input	i_source i_target	Index of the array object that should be copied. The value must be scalar, integer and greater than 0 and it can not exceed the array size. Target can be a scalar or vector with elements > 0.

par = **qd_arrayant.generate** (array_type, Ain, Bin, Cin, Din, Ein, Fin, Gin, Hin, Iin, Jin)

Calling object	None (static method)	
Description	Generates predefined array antennas	
Array types	omni dipole half-wave-dipole patch	An isotropic radiator with vertical polarization. A short dipole radiating with vertical polarization. A half-wave dipole radiating with vertical polarization. A vertically polarized patch antenna with 90° opening in azimuth and elevation.

	custom	An antenna with a custom gain in elevation and azimuth. The values A,B,C and D for the parametric antenna are returned. Ain - 3dB beam width in azimuth direction Bin - 3dB beam width in elevation direction Cin - Isotropic gain (linear scale) at the back of the antenna
	parametric	An antenna with the radiation pattern set to $$E\theta = A \cdot \sqrt{B + (1-B) \cdot (\cos\theta)^C \cdot \exp(-D \cdot \phi^2)}$$
	multi	A multi-element antenna with adjustable electric downtilt. Ain - Number of elements stacked in elevation direction Bin - Element spacing in [λ] Cin - Electric downtilt in [deg] Din - Individual element pattern "Fa" for the vertical polarization Ein - Individual element pattern "Fb" for the horizontal polarization
	3gpp-macro	An antenna with a custom gain in elevation and azimuth. See. 3GPP TR 36.814 V9.0.0 (2010-03), Table A.2.1.1-2, Page 59 Ain - Half-Power in azimuth direction (default = 70 deg) Bin - Half-Power in elevation direction (default = 10 deg) Cin - Front-to back ratio (default = 25 dB) Din - Electrical downtilt (default = 15 deg)
	3gpp-3d	The antenna model for the 3GPP-3D channel model (TR 36.873, v12.5.0, pp.17). Ain - Number of vertical elements (M) Bin - Number of horizontal elements (N) Cin - The center frequency in [Hz] Din - Polarization indicator 1. K=1, vertical polarization only 2. K=1, H/V polarized elements 3. K=1, +/-45 degree polarized elements 4. K=M, vertical polarization only 5. K=M, H/V polarized elements 6. K=M, +/-45 degree polarized elements Ein - The electric downtilt angle in [deg] for Din = 4,5,6 Fin - Element spacing in [λ], Default: 0.5
	3gpp-mmw	Antenna model for the 3GPP-mmWave channel model (TR 38.901, v14.1.0, pp.21). The parameters "Ain" - "Fin" are identical to the above model for the "3gpp-3d" channel model. Additional parameters are: Gin - Number of nested panels in a column (Mg) Hin - Number of nested panels in a row (Ng) Iin - Panel spacing in vertical direction (dg, V) in [λ], Default: 0.5 M Jin - Panel spacing in horizontal direction (dg, H) in [λ], Default: 0.5 N
	parabolic	An ideal parabolic reflector antenna with input parameters: Ain - Radius of the antenna aperture in [meters] Bin - Center frequency in [Hz] Cin - Min. sidelobe power relative to directivity in [dB] (default: -40 dB) Din - Polarization indicator 1. vertical (E-theta) polarization (default) 2. horizontal (E-phi) polarization 3. LHCP 4. RHCP 5. dual-polarized two-port antenna (LHCP, RHCP)
	xpol	Two elements with ideal isotropic patterns (vertical polarization). The second element is slanted by 90°.

	rhcp-dipole	Two crossed dipoles with one port. The signal on the second element (horizontal) is shifted by -90° out of phase. The two elements thus create a right hand circular polarized (RHCP) signal.
	lhcp-dipole	Two crossed dipoles with one port. The signal on the second element (horizontal) is shifted by 90° out of phase. The two elements thus create a left hand circular polarized (LHCP) signal.
	lhcp-rhcp-dipole	Two crossed dipoles. For input port 1, the signal on the second element is shifted by +90° out of phase. For input port 2, the the signal on the second element is shifted by -90° out of phase. Port 1 thus transmits a LHCP signal and port 2 transmits a RHCP signal.
	ula2	Uniform linear arrays composed of 2 omni-antennas (vertical polarization) with 10 cm element spacing.
	ula4	Uniform linear arrays composed of 4 omni-antennas (vertical polarization) with 10 cm element spacing.
	ula8	Uniform linear arrays composed of 8 omni-antennas (vertical polarization) with 10 cm element spacing.
	vehicular	Generates array antennas for vehicle UEs according to 3GPP TR 37.885 V15.1.0. Ain - vehicle type 1. passenger vehicle w/ bumper antennas 2. passenger vehicle w/ rooftop antennas 3. bus/truck w/ rooftop antennas Bin - frequency range 1. below 6 GHz 2. above 6 GHz Cin - model option 1. antennas based on macro BS antenna pattern 2. antenna patterns based on simulated vehicle mounted antennas
Input	array_type Ain - Jin	One of the above array types. Additional parameters for the array antenna (see above).
Output	par	The parameters A, B, C, and D for the ”parametric” antenna type.

[V, H, CP, dist] = interpolate (azimuth, elevation, i_element)

Calling object	Single object	
Description	Interpolates the field pattern Interpolation of the beam patterns is very computing intensive. It must be performed several thousands of times during a simulation run. Therefore, optimized 2D linear interpolation is used. There are additional input parameters specified in the .m-File that are not in the list below. Those parameters correspond to the properties of the '`qd_arrayant`' class. Passing those variables during the function call takes less time than reading them from the object properties. This is used internally in '`qd_builder.get_channels`' but is irrelevant here.	
Input	azimuth elevation i_element	A vector of azimuth angles in [rad] A vector of elevation angles in [rad] The element indices for which the interpolation is done. If no element index is given, the interpolation is done for all elements in the array.
Output	V H dist	The interpolated vertical field pattern (θ-component) The interpolated horizontal field pattern (ϕ-component) The effective distances between the antenna elements when seen from the direction of the incident path. The distance is calculated by an projection of the array positions on the normal plane of the incident path. This is needed for the planar wave approximation.

gain_dBi = **normalize_gain** (i_element, gain)

Calling object	Single object	
Description	Normalizes all patterns to their gain	
Input	i_element gain	A list of elements for which the normalization is done. Default: All elements The gain that should be set in the pattern. If this variable is not given, the gain is calculated from the pattern
Output	gain_dBi	Normalized gain of the antenna

rotate_pattern (deg, rotaxis, i_element, usage)

Calling object	Single object	
Description	Rotates antenna patterns Pattern rotation provides the option to assemble array antennas out of single elements. By setting the '`element_position`' property of an array object, elements can be placed at different coordinates. In order to freely design arbitrary array configurations, however, elements often need to be rotated (e.g. to assemble a +/- 45° crosspolarized array out of single dipoles). This functionality is provided here.	
Input	deg rotaxis i_element usage	The rotation angle in [degrees] ranging from -180° to 180° The rotation axis specified by the character 'x','y', or 'z'. The element indices for which the rotation is done. If no element index is given, the rotation is done for all elements in the array. The optional parameter 'usage' can limit the rotation procedure either to the pattern or polarization. Possible values are: • 0: Rotate both (pattern+polarization) - default • 1: Rotate only pattern • 2: Rotate only polarization

mse = **set_grid** (azimuth_grid, elevation_grid)

Calling object	Single object	
Description	Sets a new grid for azimuth and elevation and interpolates the pattern This function replaces the properties '`azimuth_grid`' and '`elevation_grid`' of the antenna object with the given values and interpolates the antenna patterns to the new grid.	
Input	azimuth_grid elevation_grid use_interpolate	Azimuth angles in [rad] were samples of the field patterns are provided The field patterns are given in spherical coordinates. This variable provides the azimuth sampling angles in radians ranging from $-\pi$ to π. Elevation angles in [rad] were samples of the field patterns are provided The field patterns are given in spherical coordinates. This variable provides the elevation sampling angles in radians ranging from $-\frac{\pi}{2}$ (downwards) to $\frac{\pi}{2}$ (upwards). Switch to enable (1, default) or disable (0) the antenna pattern interpolation.
Output	mse	The mean-square error in [dB] that is achieved when interpolating the array antenna with the new grid back to the original grid. Larger values are better. Values close to zero or below zero indicate that that pattern sampling interval is too low to reconstruct the phases of the original pattern. The pattern has become useless.

a = **sub_array** (i_element)

Calling object	Single object	
Description	Generates a sub-array with the given array indices This function creates a copy of the given array with only the selected elements specified in `i_element`.	
Input	i_element	A list of element indices
Output	a	An arrayant object with the desired elements

h_figures = **visualize** (i_element)

Calling object	Single object	
Description	Create a plot showing the element configurations	
Input	i_element	The element indices for which the plot os created. If no element index are given, a plot is created for each element in the array.
Output	h_figures	The figure handles for further processing of the images.

[Fa, Fb, azimuth_grid, elevation_grid, element_position] = **wrap_grid** (i_element, precision)		
Calling object	Single object	
Description	Wraps the antenna patterns around the unit sphere This function reads the antenna patterns from the `qd_arrayant` object and checks if the pattern is wrapped. This is defined as: • The first angle in the azimuth grid is smaller or equal to -pi • The last angle in the azimuth grid is larger or equal to pi • The first elevation angle is -pi/2 • The last elevation angle is pi/2 These conditions are required for the antenna pattern interpolation which can only interpolate, but not extrapolate. The output of this function are the completed patterns.	
Input	i_element	The element indices that should be returned.
	precision	If set to 'single', single precision variables are returned.
Output	Fa	The first component of the antenna pattern
	Fb	The second component of the antenna pattern
	azimuth_grid	Azimuth angles (theta) in [rad]
	elevation_grid	Elevation angles (phi) in [rad]
	element_position	Position of the antenna elements

[h_array, l] = **qd_arrayant.xml_read** (fn, fid, pfx, l, ignore_layout)		
Calling object	None (static method)	
Description	Reads antenna patterns from a QDANT XML file The QuaDRiGa array antenna exchange format (QDANT) is a file format used to store antenna pattern data in XML and load them into QuaDRiGa. The file format specification is described in the documentation. This method loads the correctly formatted XML file into a `qd_arrayant` object array.	
Input	fn	Filename of the QDANT XML file.
	fid	An integer that identifies an already opened file for subsequent low-level file I/O operations (e.g. used when loading antenna data embedded in a KML file).
	pfx	XML namespace declaration (string). It is possible to use a prefix to avoid name conflicts when embedding QuaDRiGa antennas in other XML formats. The variable "pfx" is only required if the XML file uses a namespace to identify the antenna objects.
	l	Current line of the already opened file identified by `fid`.
	ignore_layout	Boolean value. By default (0), the layout of multiple `qd_arrayant` objects is stored in the QDANT file. This layout is restored by `xml_read`. Setting `ignore_layout` to 1 loads all `qd_arrayant` objects in a [1 x N] object array.
Output	h_array	Array of `qd_arrayant` objects.
	l	Last line of an already opened file identified by `fid` that was processed by `xml_read`.

xml_write (fn, pfx, id, fid)		
Calling object	Object array	
Description	Writes antenna patterns into a QDANT XML file The QuaDRiGa array antenna exchange format (QDANT) is a file format used to store antenna pattern data in XML. The file format specification is described in the documentation. This method saves a `qd_arrayant` object array to a XML file.	
Input	fn	Filename of the QDANT XML file.
	pfx	String defining the namespace (optional). It is possible to use a prefix to avoid name conflicts when embedding QuaDRiGa antennas in other XML formats. When using a prefix in XML, a namespace for the prefix must be defined.
	id	Integer number defining the array antenna ID (optional). If multiple array antennas are stored in the same file, each antenna must be identified by an unique ID.
	fid	An integer that identifies an already opened file for subsequent low-level file I/O operations (optional).

2.2.3 Class "qd_track"

One feature of the channel model is the continuous evolution of wireless channels when the terminal moves through the environment. A track describes the movement of a mobile terminal. It is composed of an ordered list of positions. During the simulation, one snapshot is generated for each position on the track.

Along the track, wireless reception conditions may change, e.g. when moving from an area with LOS to a shaded area. This behavior is described by segments, or states. A segment is a subset of positions that have similar reception conditions. Each segment is classified by a segment index (i.e. the center position of the segment) and a scenario.

Properties

name	Name of the track
no_snapshots	Number of positions on the track
initial_position	Position offset (will be added to positions) This position is given in global cartesian coordinates (x,y, and z-component) in units of [m]. The initial position normally refers to the starting point of the track. If the track has only one segment, it is also the position for which the LSPs are calculated. The initial position is added to the values in the positions variable.
positions	Ordered list of positions relative to the initial position QuaDRiGa calculates an instantaneous channel impulse response (also called snapshot) for each position on the track.
orientation	This 3-element vector describes the orientation of the radio device for each position on the track. The reference system for aircraft principal axes is used. The first value describes the "bank angle", i.e. the orientation around an axis drawn through the body of the vehicle from tail to nose in the normal direction of movement. Positive rotation is clockwise (seen from the pilot/drivers perspective). The second value describes the "tilt angle", i.e. the vertical angle relative to the horizontal plane; positive values point upwards. The third value describes the bearing or "heading angle", in mathematic sense. Heading is used to describe the direction an object is pointing. In contrast, the course angle refers to the direction an object is actually moving. East corresponds to 0, and the angles increase counter-clockwise, so north is 90 degree degrees, south is -90 degree, and west is 180 degree. All values are given in [rad]. Note that by default, QuaDRiGa antennas face east (bank = 0, tilt = 0, heading = 0).

movement_profile	Time (in sec) vs. distance (in m) for speed profile
	QuaDRiGa supports variable terminal speeds. This is realized by interpolating the channel coefficients at the output of the model. The variable '`track.movement_profile`' describes the movement along the track by associating a time-point with a distance-point on the track. An example is:
	`t.movement_profile = [ 0,7 ; 5,0 ; 6,0 ; 20,20 ]';`
	`dist = t.interpolate_movement( 1e-3 );`
	`ci = cn.interpolate( dist , t.get_length );`
	See also the tutorial "Applying Varying Speeds (Channel Interpolation)" for more details.

no_segments	Number of segments or states along the track
segment_index	Starting point of each segment given as index in the '`positions`' vector
scenario	Scenarios for each segment along the track This variable contains the scenario names for each segment as a cell array of strings. A list of supported scenarios can be obtained by calling '`qd_builder.supported_scenarios`'. If there is only one transmitter (i.e. one base station), the cell array has the dimension [1 x no_segments]. For multiple transmitters, the rows of the array may contain different scenarios for each transmitter. For example, in a multicell setup with three terrestrial base stations, the propagation conditions may be different to all BSs. Hence, cell arrays have a dimension [3 x no_segments].
par	Field for storing additional data
closed	Indicates that the track is a closed curve

Methods

h_track = **qd_track** (track_type, varargin)	
Calling object	None (constructor)
Description	Creates a new track object The constructor calls `qd_track.generate` to create new tracks. If no input is specified, a linear track with 1 m length is generated. See 'qd_track.generate' for a description of the input parameters and the list of supported antenna types.

d_min = **add_segment** (pos, scenario, threshold)		
Calling object	Object array	
Description	Adds segments to a qd_track object of array of objects This method can be used to add segments to an existing `qd_track` object or array of `qd_track` objects. The variable `pos` defines the position where the segment should start. Ideally, this positions lies on a track. However, if it doesn't lie on a track, the closest point on the nearest track is used. If the track has no point at this position, a new one will be created. For example, a 100 m long linear track is defined by its start and end point (the `qd_track` object contains two positions, 0 and 100 meters). If a new segment should start at 50 m relative to the track start, the `add_segment` method will create a new point at 50 m and assign the segment to it. The `qd_track` object will then contain 3 points (0 m, 50 m and 100 m).	
Input	pos	A 3-element vector [x;y;z] in metric Cartesian coordinates defining the start-position of the new segment (required).
	scenario	A string or cell-array of strings providing the scenario name (optional). Scenario names are defined by the configuration files in the `config` folder of the QuaDRiGa installation. A list of supported scenarios can be obtained by calling "`qd_builder.supported_scenarios`". The scenario cell-array can only have one column. Rows are for different transmitters. If scenario is not defined, no new points are added to the track, but the distances are calculated.
	threshold	A scalar value in meters describing the minimum distance to an existing point on a track at which no new point is created, but the scenario is assigned to the existing point (optional). The default value is 0.1 meters.
Output	d_min	An array of floating point numbers describing the distance to each track in the array of `qd_track` objects. The scenario is assigned to the object with the minimum distance.

calc_orientation	
Calling object	Object array
Description	Calculates the orientation of the mobile device from the positions This function calculates the orientations of the mobile device based on the snapshot positions. If we assume that the receive array antenna is fixed on a car and the car moves along the track, then the antenna turns with the car when the car is changing direction. This needs to be accounted for when generating the channel coefficients. This function calculates the orientation based on the positions and stores the output in the `orientation` property of the track object. The 3-element orientation vector describes the orientation of the radio device for each position on the track. The reference system for aircraft principal axes is used. The first value describes the "roll angle", i.e. the rotation around an axis drawn through the body of the vehicle from tail to nose in the normal direction of movement. Positive rotation is clockwise (seen from the pilot/drivers perspective). This values is set to 0 when using "`calc_orientation`". The second value describes the "pitch angle", i.e. the vertical (tilt) angle relative to the horizontal plane; positive rotation is up. The third value describes the bearing or "yaw angle", in mathematic sense. East corresponds to 0, and the angles increase counter-clockwise, so north is 90 degree degrees, south is -90 degree, and west is 180 degree. All values are given in [rad]. Note that by default, QuaDRiGa antennas face east (roll = 0, pitch = 0, yaw = 0).

out = **copy**		
Calling object	Object array	
Description	Creates a copy of the handle class object or array of objects. While the standard copy command creates new physical objects for each element of the object array (in case obj is an array of object handles), copy checks whether there are object handles pointing to the same object and keeps this information.	
Output	out	Copy of the current object or object array

correct_overlap (overlap)		
Calling object	Object array	
Description	Corrects positions of the segment start to account for the overlap. After the channel coefficients are calculated, adjacent segments can be merged into a time-continuous output. The merger assumes that the merging interval happens at the end of one segment, before a new segments starts. In reality, however, the scenario change happens in the middle of the overlapping part (and not at the end of it). This function corrects the position of the segment start to account for that.	
Input	overlap	The length of the overlapping part relative to the segment length. It can have values in between 0 (no overlap) and 1 (ramp along the entire segment). The default value is 0.5. You need to make sure that the same value is used when calling "qd_channel.merge".

h_track = **qd_track.generate** (track_type, track_length, direction, street_length_min, street_length_mu, street_length_std, curve_radius, turn_probability)		
Calling object	None (static method)	
Description	Generate tracks This function creates tracks with specific properties. Currently supported are "linear", "circular" and "street".	
Track types	linear	Creates a linear track with given length and direction. Direction describes the travel direction along the track in [rad] in mathematical sense (i.e. 0 means east, pi/2 means north, pi means west and -pi/2 south). If "track_length" or "direction" is not specified, then the default track is 1 m long and has a random direction.
	circular	Creates a circular track with given length and starting-direction. Direction defines the starting point on the circle in [rad]. Positive values define the travel direction as counter clock-wise and negative values as clock-wise. E.g. 0 sets the start point in the east of the circle, traveling north; -2π sets it in the east, traveling south. The default is random.
	street	Emulates a drive route through a city grid. The mobile terminal starts at point 0, going into a specified direction. The trajectory grid is build from street segments. The length of each street is specified by the parameters '`street_length_min`', '`street_length_mu`', and '`street_length_sigma`'. At the end of a street (i.e. at a crossing), the terminal turns with a probability specified by '`turn_probability`'. The change of direction is in between 75 and 105 degrees either left or right. The radius of the curve is given by '`curve_radius`'. The track is set up in a way that prevents driving in circles.
Input	track_type	The type of the track
	track_length	the length in [m]
	direction	specifies the driving direction in [rad] of the first segment in mathematical sense (0 means east, pi/2 means north). The default value is random
	street_length_min	the minimal street length in [m]. The default is 50 m. (for type "street" only)
	street_length_mu	the median street length in [m]. The default is 187 m. This value was obtained from measurements in Berlin, Germany. (for type "street" only)
	street_length_std	the standard deviation of the street length in [m]. The default is 83 m. This value was obtained from measurements in Berlin, Germany. (for type "street" only)
	curve_radius	the curve radius during a turn in [m]. The default is 10 m. (for type "street" only)
	turn_probability	the probability of a turn at a crossing. Possible values are in between 0 and 1. The default is 0.5. (for type "street" only)
Output	h_track	A `qd_track` object

[len, dist] = **get_length**		
Calling object	Object array	
Description	Calculates the length of the track in [m]	
Output	len	Length of a track in [m]
	dist	Distance of each position (snapshot) from the start of the track in [m]

subtracks = **get_subtrack** (i_segment, i_tx)		
Calling object	Single object	
Description	Splits the track in subtracks for each segment After defining segments along the track, one needs the subtrack that corresponds only to one segment to perform the channel calculation. This new track can consist of two segments. The first segment contains the positions from the previous segment, the second from the current. This is needed to generate overlapping channel segments for the merging process. This function returns the subtracks for the given segment indices. When no input argument is provided, all subtracks are returned.	
Input	i_segment	A list of indices indicating which subtracks should be returned. By default, all subtracks are returned.
Input	i_tx	A list of indices indicating which transmitter should be returned. Usually, each transmitter in a layout gets assigned a scenario. The scenario-IDs are stored in "qd_track.scenario", where the number of rows corresponds to the number of transmitters in a layout. By default, all transmitters are returned.
Output	subtracks	A vector of `qd_track` objects corresponding to the number of segments.

[dist, h_track_int] = **interpolate** (method, sample_rate, movement_profile, algorithm, update_input)		
Calling object	Single object	
Description	Interpolates the snapshot positions along the track This function interpolates the positions along the track such that it matches the given sample rate. The channel model operates on a position-based sample grid. That means that the '`builder`' generates one CIR for each position on the track. In practise, however, a time continuous evolution of the CIR is often needed. This can be interpolated from the position-based grid, provided that the spatial sample theorem is not violated (i.e. the channel needs to be sampled at least twice per half wave length). In order to do that, enough sample points are needed along the track.	
Interpolation method	distance	The distance-based interpolation calculates the missing sample points and places them equally spaced along the track. This corresponds to a constant speed when evaluating the output CIRs. The required minimum value for '`sample_rate`' is equal to '1/`samples_per_meter`' from the '`qd_simulation_parameters`' object.
	time	The time-based interpolation is needed in dual-mobility scenarios, where Tx and Rx can move at different speeds. In this case the sample rate is given in [seconds] and a movement profile must be given. If you use the `set_speed` function of the `qd_track` object, the movement profile is automatically generated.
	snapshot	Time-based interpolation where positions on the track are given by snapshot numbers instead of distances. This is useful when the device is stationary but changes it's orientation over time, e.g. a radar antenna. The sample rate must be given in [seconds]. The movement profile must contain the time-points (first row) vs. snapshot number (second row). Snapshot indices start from 1. Fractional number are possible (e.g. 1.5).
Input	method	Must be either '`distance`', '`time`' or '`snapshot`'.
	sample_rate	The interval between two (output) snapshots given in [meters] for distance-based interpolation and [seconds] for time-based and snapshot-based interpolation.
	movement_profile	A matrix describing the time (in seconds) vs. distance (in meters) for time-based interpolation; or (input) snapshot-number vs. distance for snapshot-based interpolation. The first row describes the time points, the second row describes the positions on the track relative to the start point.
	algorithm	Selects the interpolation algorithm. Optional are • linear - Linear interpolation (default) • cubic - Shape preserving piecewise cubic interpolation Interpolation of the orientations are always done using the SLERP algorithm (spherical linear interpolation).

	update_input	When set to 'true', the input track object is updated. If set to 'false' (default), a new track object is created.
Output	dist	The interpolated distances relative to the track start point.
	h_track_int	A new 'qd_track' object containing the interpolated positions, orientations, segments and parameters. If 'update_input' is set to true, the calling track handle is returned.

dist = **interpolate_movement** (si, algorithm)

Calling object	Single object	
Description	Interpolates the movement profile to a distance vector This function interpolates the movement profile. The distance vector at the output can then be used to interpolate the channel coefficients to emulate varying speeds. See also the tutorial "Applying Varying Speeds (Channel Interpolation)".	
Input	si	the sampling interval in [seconds]
	algorithm	selects the interpolation algorithm. The default is cubic spline interpolation. Optional are: • linear - Linear interpolation (default) • cubic - Shape preserving piecewise cubic interpolation
Output	dist	Distance of each interpolated position from the start of the track in [m]

interpolate_positions (samples_per_meter)

Calling object	Object array	
Description	Interpolates positions along the track This function interpolates the positions along the track such that it matches the samples per meter specifies in the simulation parameters. The channel model operates on a position-based sample grid. That means that the 'builder' generates one CIR for each position on the track. In practise, however, a time continuous evolution of the CIR is often needed. This can be interpolated from the position-based grid, provided that the spatial sample theorem is not violated (i.e. the channel needs to be sampled at least twice per half wave length). In order to do that, enough sample points are needed along the track. This function calculates the missing sample points and places them equally spaced along the track. This corresponds to a constant speed when evaluating the output CIRs. The required value for 'samples_per_meter' can be obtained from the 'qd_simulation_parameters' object.	
Input	samples_per_meter	the samples per meter (e.g. from 'qd_simulation_parameters.samples_per_meter')

set_scenario (scenario, probability, seg_length_min, seg_length_mu, seg_length_std)

Calling object	Object array	
Description	Assigns random scenarios and creates segments. This function can be used to create segments along the trajectory and assign scenarios to the segments. If there are less than 3 input arguments (i.e. only 'scenario' and/or 'probability' is given), then no segments will be created. To create segments with the default settings call 'set_scenario(scenario,[],[])'. Alternatively, it is possible to only create segments by leaving the scenario empty, e.g. by calling 'set_scenario([],[],[])'.	
Input	scenario	A cell array of scenario-names. Each scenario (synonym for propagation environment) is described by a string (e.g. "MIMOSA_16-25_LOS" or "WINNER_SMa_C1_NLOS"). A list of supported scenarios can be obtained by calling 'parameter_set.supported_scenarios'. The scenario parameters are stored in the configuration folder "config" in the QuaDRiGa main folder. The filenames (e.g. "MIMOSA_16-25_LOS.conf") also serves as scenario name.
	probability	The probability for which the scenario occurs. This parameter must be a vector of the same length as there are scenarios. Probabilities must be specified in between 0 and 1. The sum of the probabilities must be 1. By default (or when 'probability' is set to '[]'), each scenario is equally likely.
	seg_length_min	the minimal segment length in [m]. The default is 10 m.
	seg_length_mu	the median segment length in [m]. The default is 30 m.
	seg_length_std	the standard deviation of the street length in [m]. The default is 12 m.

set_speed (speed)		
Calling object	Object array	
Description	Sets a constant speed in [m/s] for the entire track. This function fills the '`qd_track.movement_profile`' field with a constant speed value. The movement profile describes the movement along the track by associating a time-point with a distance-point on the track. If the track length is 0 (e.g. for static transceivers), the first row of the movement profile will contain time in seconds (calculated from 1/speed) and the second row will contain the snapshot number.	
Input	speed	The terminal speed in [m/s]

split_segment (mi, ma, mu, sig, no_check)		
Calling object	Object array	
Description	Splits long segments in subsegments of the same type.	
Input	mi	Minimum length of the subsegment in [m], default: 10m
	ma	Maximum length of the subsegment in [m], must be > 2*mi, default: 30m
	mu	Mean length of the subsegment (mi < mu < ma), default: 15m
	sig	Std of the length of the subsegment, default: 5m
	no_check	Disable parsing of input variables, default: false

2.2.4 Class "qd_layout"

Objects of this class define the network layout of a simulation run. Each network layout has one or more transmitters and one or more receivers. Each transmitter and each receiver need to be equipped with an array antenna which is defined by the `qd_arrayant` class. In general, it is assumed that the transmitter is at a fixed position and the receiver is mobile. Thus, each receivers movement is described by a track.

Properties

name	Name of the layout
simpar	Handle of a 'qd_simulation_parameters' object. See Section 2.2.1
update_rate	Channel update rate in seconds
no_tx	Number of transmitters (or base stations)
no_rx	Number of receivers (or mobile terminals)
tx_name	Identifier of each Tx, must be unique
tx_position	Position of each Tx in global cartesian coordinates using units of [m]
tx_array	Handles of 'array' objects for each Tx. See Section 2.2.2
rx_name	Identifier of each Tx, must be unique
rx_position	Initial position of each Rx (relative to track start) in global cartesian coordinates using units of [m]
rx_array	Handles of 'array' objects for each Rx. See Section 2.2.2
track	Handles of track objects for each Rx. See Section 2.2.3
pairing	An index-list of links for which channels are created. The first row corresponds to the Tx and the second row to the Rx.
no_links	Number of links for which channel coefficients are created (read only)
dual_mobility	Indicator if the layout contains moving transmitters

Methods

h_layout = **qd_layout** (simpar)		
Calling object	None (constructor)	
Description	Creates a new `qd_layout` object.	
Input	simpar	Handle of a '`qd_simulation_parameters`' object.

out = **copy**		
Calling object	Object array	
Description	Creates a copy of the handle class object or array of objects. While the standard copy command creates new physical objects for each element of the object array (in case obj is an array of object handles), copy checks whether there are object handles pointing to the same object and keeps this information.	
Output	out	Copy of the current object or object array

o2i_loss_dB = **gen_o2i_loss** (low_loss_fraction, SC_lambda, max_indoor_distance)		
Calling object	Single object	
Description	Generates the outdoor-to-indoor penetration loss for the 3GPP 38.901 model This method generates the outdoor-to-indoor (O2I) penetration loss in [dB] for the 3GPP 38.901 channel model. The O2I-loss is specific for each terminal. In other words, if a MT is served by several BSs, the same O2I-loss applies for each BSs. The values therefore need to be generated before any other LSF or SSF parameters are generated. The automatic channel generation in '`qd_layout.get_channels`' will call all functions in the correct order. For details see: 3GPP TR 38.901 v14.1.0, Sec. 7.4.3, Page 27. The method generates the O2I-loss and indoor 3D distance as specified by 3GPP. The values are then stored with the tracks in '`qd_layout.track.par.o2i_loss`' and '`qd_layout.track.par.o2i_d3din`'. These two variables are then read again by '`qd_builder.get_pl`' and applied to the path-loss in the specific scenario.	
Input	low_loss_fraction	3GPP TR 38.901 specifies two different formulas for the O2I-loss, one for high-loss (e.g. IRR glass and concrete) and one for low-loss (standard multi-pane glass). The variable '`low_loss_fraction`' determines the likelihood of the low-loss model. Values must be between 0 (high-loss only) and 1 (low-loss only). Default-value is 0.5.
	SC_lambda	Random variables for the '`low_loss_fraction`' are spatially consistent. '`SC_lambda`' describes the decorrelation distance of the random generator. Default: 50 m.
	max_indoor_dist	The maximum indoor distance between the building wall and the UE. Default is 25 m.
Output	o2i_loss_dB	A cell array containing the results for each MT. These values are identical to the ones stored in '`qd_layout.track.par.o2i_loss`'. Each cell contains an array of values, where the dimensions correspond to: [BS, Segment, Frequency]. For outdoor-scenarios, an empty array is returned.

h_layout = **qd_layout.generate** (layout_type, no_sites, isd, h_arrayant, no_sectors, sec_orientation)		
Calling object	None (static method)	
Description	Generates predefined network layouts.	
Layout types	hexagonal	A hexagonal network layout with up to three rings of BSs. The first BS is at coordinates [0,0]. The first BS of each ring is placed in the (north)-east of BS1. Additional BSs are added in mathematical positive sense (counter-clockwise). Default BS height is 25 m.
	regular	Same as hexagonal. However, each BS has three sectors. The number of sites can be 1, 7, 19 or 37 - resulting in 3, 21, 57 or 111 sectors, respectively. Sector orientations are 30, 150 and -90 degrees in mathematical sense.
	regular6	Same as hexagonal, but with default settings of 6 sectors per site and sector orientations 0, 60, 120, 180, 240 and 300 degree.
	indoor	3GPP 38.901 indoor scenario. The number of sites can be given by a two-element array [N, M], where N denotes the number of BSs in y-direction and m in x-direction. Default BS height is 3 m.
	random	Randomly places base stations. The input parameter ISD describes the radius of the layout in [m].
Input	layout_type	The layout type (string)
	no_sites	The number of BS sites in the layout.
	isd	The inter-site distance in [m]
	h_arrayant	The array antenna object for each sector.
	no_sectors	The number of sectors per site (default: 1)
	sec_orientation	The orientation offset of the first sector in [deg]. (default: 0 deg)
Output	h_layout	The generated layout

[h_channel, h_builder] = **get_channels** (update_rate, check_parfiles, overlap, algorithm)		
Calling object	Single object	
Description	Generate the channel coefficients. This method executes all necessary functions to generate the channel coefficients. These are: 1. Interpolation of the tracks to match the sample density. This avoids unnecessary computations. The minimum sample density (`qd_layout.simpar.sample_density`) is 1 sample per half-wavelength for static transmitters and 2 samples per half-wavelength for mobile transmitters (dual-mobility). The default value is 2.5. The interpolation is only done if a sample rate is provided. 2. Generation of channel builder objects and assigning track segments to builders. 3. Generation of large and small-scale-fading parameters, including spatial consistency. 4. Splitting of builder object for multi-frequency simulations (only when multiple carrier frequencies are given in (`qd_layout.simpar.center_frequency`). 5. Generation of drifting channel coefficients for each track-segment. 6. Merging of channel segments, including modeling the birth and death of scattering clusters. 7. Interpolation of channel coefficients to match the sample rate (only if sample rate is provided). 8. Formatting of output channel objects to an object array with dimensions: [Rx , Tx , Freq.] If `qd_layout.simpar.show_progress_bars` is set to 1, a progress report is provided on the command prompt.	
Input	update_rate	Channel update rate in [s]. If this parameter is omitted or empty, the value from the class property "qd_layout.update_rate" is used. If this value is empty, no track and channel interpolation is performed and channel coefficients are generated for each snapshot specified in the track objects. If the parameter is given, each track must have a movement-profile.
	check_parfiles	check_parfiles = 0 / 1 (default: 1) Disables (0) or enables (1) the parsing of shortnames and the validity-check for the config-files. This is useful, if you know that the parameters in the files are valid. In this case, this saves execution time.
	overlap	The length of the overlapping part relative to the segment length (segments are specified in `qd_layout.rx_track`. It can have values in between 0 (no overlap) and 1 (ramp along the entire segment).
	algorithm	Selects the interpolation algorithm for the tracks and channel coefficients. Optional are • linear - Linear interpolation (default) • cubic - Shape preserving piecewise cubic interpolation Interpolation of the transceiver orientations and channel phases are always done using the SLERP algorithm (spherical linear interpolation). Spherical cubic interpolation is currently not supported.
Output	h_channel h_builder	A vector '`qd_channel`' objects. A vector of '`qd_builder`' objects.

h_builder = **init_builder** (check_parfiles, split_tx)

Calling object	Single object	
Description	Creates 'qd_builder' objects based on layout specification This function processes the data in the 'qd_layout' object. First, all tracks in the layout are split into subtracks. Each subtrack corresponds to one segment. Then, then scenario names are parsed. A 'qd_builder' object is created for each scenario and for each transmitter. For example, if there are two BSs, each having urban LOS and NLOS users, then 4 'qd_builder' objects will be created (BS1-LOS, BS2-NLOS, BS2-LOS, and BS2-NLOS). The segments are then assigned to the 'qd_builder' objects.	
Input	check_parfiles	Enables (1, default) or disables (0) the parsing of shortnames and the validity-check for the config-files. This is useful, if you know that the parameters in the files are valid. In this case, this saves some execution time.
	split_tx	If set to true (1), each Tx gets assigned to a new builder object. Hence, all LSPs and SSF parameters will be independently generated for each Tx. If set to false (0), Txs belonging to the same scenario will be combined into one builder, enabling spatial consistency for the Txs. The default value is 1, if all Txs are static, and 0, if at least on Tx is mobile (dual-mobility feature).
Output	h_builder	A matrix of 'qd_builder' objects. Rows correspond to the scenarios, columns correspond to the transmitters.

[h_layout, ReferenceCoord] = **qd_layout.kml2layout** (fn, split_seg)

Calling object	None (static method)	
Description	Imports a layout object from a KML file This function loads a QuaDRiGa layout from a KML-File. KML-Files are created e.g. by Google(TM) maps or a GPS device. In the KML-File, the user can specify the positions of the transmitters (Tx), the receiver tracks (Rx) and segments for the receiver track. In order to work properly, the KML-File needs to meet some specific formatting requirements: Tx-positions are represented by a "Placemark" in the KML-file. The name must be "tx_TxName" where 'TxName' has to be replaced by a unique name for each transmitter. Rx-Tracks are specified by paths. Each path found in the KML-File is interpreted as a Rx-track. As for the Tx, all paths must contain a unique name. Segments of a Rx-Track are determined by placemarks in close proximity to the track. The name of the placemark contains the scenario. The naming convention for segments is "seg_Scen" where 'Scen' determines the scenaio of the segment. For example, you can create a path with the name "rx_GPS1". In order so assign the scenario "WINNER_UMa_C2_LOS" to the path, you need to add a placemark at the beginning of the track with the name: "seg_WINNER_UMa_C2_LOS". It also possible to specify a different scenario for each transmitter in the layout. The naming convention then is "seg_ScenTx1:ScenTx2:...:ScenTxN" where the scenarios for each transmitter are separated by a ":". A complete description of QuaDRiGa-KML specification can be found in the documentation.	
Input	fn	The filename of the KML-File (string)
	split_seg	It set to true (1, default), tracks are split into segment as indicated by the parameter "SplitSegments" in the KML file. If set to false (0), "SplitSegments" is ignored.
Output	h_layout	The 'qd_layout' object
	ReferenceCoord	A tuple for longitude and latitude (WGS84) at which the origin (0,0,0) of the metric Cartesian coordinate system used by QuaDRiGa is placed.

layout2kml (fn, reference_coord, embed_antennas, use_description, split_segments)

Calling object	Single object	
Description	Exports a layout object to a KML file This function exports a QuaDRiGa layout object to a KML file. KML-Files can be read e.g. by Google(TM) maps. A complete description of QuaDRiGa-KML specification can be found in the documentation.	
Input	fn	The filename of the KML-File.
	reference_coord	A tuple for longitude and latitude (WGS84) at which the origin (0,0,0) of the metric Cartesian coordinate system used by QuaDRiGa is placed (optional). If this value is not given, the origin is placed at QuaDRiGas origin: 13.324947e,52.516319n.
	embed_antennas	Boolean value (optional). By default (1), antennas are embedded into the KML file. If disabled (0), antennas are written to an external QDANT file.

	use_description	Boolean value (optional). By default (0), additional QuaDRiGa simulation parameters are written to the `ExtendedData` elements in the KML file. If enabled (1), parameters are written to the `description` element. Description elements can be edited in Google earth.
	split_segments	This parameter controls the splitting of long segments into sub-segments with the same scenario definition. `SplitSegments` is a tuple of 4 values defining: 1. min. length of a sub-segment (e.g. 10 m) 2. max. length of the sub-segment; must be > 2·min. (e.g. 30 m) 3. average length of the sub-segment (e.g. 15 m) 4. standard-deviation of a sub-segment (e.g. 5 m) The four values are written to the KML file and applied when the file is loaded by `kml2layout`. The effect will not be visible when viewing the KML file in Google earth. If `SplitSegments` is not defined, segment splitting is disabled.

[map, x_coords, y_coords] = **power_map** (scenario, usage, sample_distance, x_min, x_max, y_min, y_max, rx_height, tx_power)

Calling object	Single object	
Description	Calculates a power-map for the given layout. This function calculates receive power values in [W] on a square lattice at a height of 'rx_height' above the ground for the given layout. This helps to predict the performance for a given setup.	
Input	scenario	The scenario for which the map shall be created. There are four options: 1. A string describing the scenario. A list of supported scenarios can be obtained by calling '`qd_builder.supported_scenarios`'. 2. cell array of strings describing the scenario for each transmitter in the layout. 3. A '`qd_builder`' object. This method is useful if you need to edit the parameters first. For example: call '`p = qd_builder('UMal')`' to load the parameters. Then edit '`p.scenpar`' or '`p.plpar`' to adjust the settings. 4. An array of '`qd_builder`' objects describing the scenario for each transmitter in the layout.
	usage	A string specifying the detail level. The following options are implemented: • '`quick`' - Uses the antenna patterns, the LOS path, and the path gain from the scenario • '`sf`' - Uses the antenna patterns, the LOS path, the path gain from the scenario, and a shadow fading map • '`detailed`' - Runs a full simulation for each pixel of the map (very slow) • '`phase`' - Same as quick, but the output contains the complex-valued amplitude instead of the power
	sample_distance	Distance between sample points in [m] (default = 10 m)
	x_min	x-coordinate in [m] of the top left corner
	x_max	x-coordinate in [m] of the bottom right corner
	y_min	y-coordinate in [m] of the bottom right corner
	y_max	y-coordinate in [m] of the top left corner
	rx_height	Height of the receiver points in [m] (default = 1.5 m)
	tx_power	A vector of tx-powers in [dBm] for each transmitter in the layout. This power is applied to each transmit antenna in the tx-array antenna. By default (if `tx_power` is not given), 0 dBm are assumed.
Output	map	A cell array containing the power map for each tx array in the layout. The power maps are given in [W] and have the dimensions [n_y_coords , n_x_coords , n_rx_elements , n_tx_elements].
	x_coords	Vector with the x-coordinates of the map in [m]
	y_coords	Vector with the y-coordinates of the map in [m]

randomize_rx_positions (max_dist, min_height, max_height, track_length, rx_ind)		
Calling object	Single object	
Description	Generates random Rx positions and tracks. Places the users in the layout at random positions. Each user will be assigned a linear track with random direction. The random height of the user terminal will be in between '`min_height`' and '`max_height`'.	
Input	max_dist	the maximum distance from the layout center in [m]. Default is 50 m.
	min_height	the minimum user height in [m]. Default is 1.5 m.
	max_height	the maximum user height in [m]. Default is 1.5 m.
	track_length	the length of the linear track in [m]. Default is 1 m.
	rx_ind	a vector containing the receiver indices for which the positions should be generated. Default: All receivers

[pairs, power] = **set_pairing** (method, threshold, tx_power, overlap, check_parfiles)		
Calling object	Single object	
Description	Determines links for which channel coefficient are generated. This function can be used to automatically determine the links for which channel coefficients should be generated. For example, in a large network there are multiple base stations and mobile terminals. The base stations, however, only serve a small area. It the terminal is far away from this area, it will receive only noise from this particular BS. In this case, the channel coefficients will have very little power and do not need to be calculated. Disabling those links can reduce the computation time and the storage requirements for the channel coefficients significantly. There are several methods to do this which can be selected by the input variable '`method`'.	
Methods	'all'	Enables the simulation of all links. Links where the Tx and the Rx are at the same position are are deactivated.
	'power'	Calculates the expected received power taking into account the path loss, the antenna patterns, the LOS polarization, and the receiver orientation. If the power of a link is below the '`threshold`', it gets deactivated.
Input	method	Link selection method. Supported are: 'all' or 'power' (see above)
	threshold	If the Rx-power is below the threshold in [dBm], the link gets deactivated
	tx_power	A vector of tx-powers in [dBm] for each transmitter in the layout. This power is applied to each transmit antenna in the tx-array antenna. By default (if '`tx_power`' is not given), 0 dBm is assumed
	overlap	The length of the overlapping part relative to the segment length. It can have values in between 0 (no overlap) and 1 (ramp along the entire segment). The default value is 0.5. You need to make sure that the same value is used when calling "qd_channel.merge". (only used for 'sf')
	check_parfiles	Disables (0) or enables (1, default) the parsing of shortnames and the validity-check for the config-files. This is useful, if you know that the parameters in the files are valid. In this case, this saves execution time.
Output	pairs	An index-list of links for which channels are created. The first row corresponds to the Tx and the second row to the Rx. An identical copy gets assigned to '`qd_layout.pairing`'.
	power	A matrix containing the estimated receive powers for each link in [dBm]. Rows correspond to the receiving terminal, columns correspond to the transmitter station. For MIMO links, the power of the strongest MIMO sublink is reported.

pos = **set_satellite_pos** (rx_latitude, sat_el, sat_az, sat_height, tx_no)		
Calling object	Single object	
Description	Calculates the Tx position from a satellite orbit. QuaDRiGas reference coordinate system is on the surface of the earth. In order to use QuaDRiGa for satellite links, the satellite position must be set. Normally, this position is given in azimuth and elevation relative to the users position. This function takes a satellite orbital position and calculates the corresponding transmitter coordinates.	
Input	rx_latitude	The receiver latitude coordinate on the earth surface in [deg]. Default is 52.5
	sat_el	Satellite elevation seen from the receiver positions in [deg]. Default is 31.6
	sat_az	Satellite azimuth in [deg] given in compass coordinates. Default is 180° (south)
	sat_height	Satellite height in [km] relative to earth surface. Default is 35786 (GEO orbit)
	tx_no	The 'tx_no' in the layout object for which the position should be set. Default is 1
Output	pos	The satellite positions in the metric QuaDRiGa coordinate system

indoor_rx = **set_scenario** (scenario, rx, tx, indoor_frc, SC_lambda_rx, SC_lambda_tx)		
Calling object	Single object	
Description	Assigns scenarios to tracks and segments. This function can be used to assign scenarios to tracks and segments of tracks. This takes the distance-dependent LOS probability into account for some specific scenarios. Currently, distance-dependent scenario selection is available for: • 3GPP_3D_UMi • 3GPP_3D_UMa • mmMAGIC_initial_UMi • mmMAGIC_initial_Indoor • 3GPP_38.901_UMi • 3GPP_38.901_UMa • 3GPP_38.901_RMa • 3GPP_38.901_Indoor_Mixed_Office • 3GPP_38.901_Indoor_Open_Office • 3GPP_38.881_DenseUrban (Satellite) • mmMAGIC_UMi • mmMAGIC_Indoor • QuaDRiGa_Industrial • QuaDRiGa_UD2D Alternatively, you can use all scenarios specified in '`qd_builder.supported_scenarios`'.	
Input	scenario	A string containing the scenario name
	rx	A vector containing the receiver indices for which the scenarios should be set. Default: all receivers
	tx	A vector containing the transmitter indices for which the scenarios should be set. Default: all transmitters
	indoor_frc	The fraction of the users (number between 0 and 1) that are indoors
	SC_lambda_rx	Decorrelation distance for spatially consistent LOS states based on the RX position. It set to 0, LOS sates are uncorrelated with respect to RX position. The default setting depends on the values of '`use_3GPP_baseline`' in the simulation setting. For '`use_3GPP_baseline = 1`', spatially consistent LOS states are uncorrelated otherwise, the values set as defined in 3GPP TR 38.901, Table 7.6.3.1-2.
	SC_lambda_tx	Decorrelation distance for spatially consistent LOS states based on the TX position. It set to 0, LOS sates are uncorrelated with respect to TX position.
Output	indoor_rx	A logical vector indicating if a user is indoors (1) or outdoors (0)

han = **visualize** (tx , rx, show_names , create_new_figure)		
Calling object	Single object	
Description	Plots the layout.	
Input	tx	A vector containing the transmitter indices that should be shown. Default: All
	rx	A vector containing the receiver indices that should be shown. Default: All
	show_names	Options: (0) shows no Tx and Rx names; (1, default) shows the Tx name and the scenario for each track segment; (2) shows the Tx and Rx name
	create_new_figure	If set to 0, no new figure is created, but the layout is plotted in the currently active figure
Output	han	The figure handle

2.2.5 Class "qd_builder"

This class implements all functions that are necessary to generate and manage correlated LSPs, correlated SSFs, and functions that are needed to generate the channel coefficients. It thus implements the core components of the channel model. The class holds all the input variables as properties. Its main function '`get_channels`' then generates the coefficients.

LSPs are the shadow fading, the Ricean K-Factor, the RMS delay spread and the four angles (elevation and azimuth at the transmitter and receiver). This class implements functions of the channel model that the user does normally not need to interact with it. However, if parameter tables need to be changed, here is the place to do so.

When calling `get_channels`, the builder generates a set of random clusters around each receiver. This is done by drawing random spatially correlated variables for the delay, the power and the departure and arrival angles for each cluster. Each cluster thus represents the origin of a reflected (and scattered) signal. The clusters are then represented as taps in the final CIR. The random variables fit the distributions and correlations defined by the LSF parameters. Spatial correlation is implemented by using the sum-of-sinuoids method for all random variables in the model.

Next, antenna dependent parameters are extracted for each user. Those depend on the position of the terminal, its orientation and the equipped antennas. The polarization rotation of the NLOS taps is modeled by a random variable which fits to the distribution defined by the LSPs. The LOS polarization is calculated from the geometric orientation of the antennas. A core function here is the interpolation of the antenna patterns which results in a specific antenna response for each subpath.

The core function then generates the coefficients themselves. This is done for each antenna element and for each snapshot separately and also includes the Doppler shift of each subpath. Finally, the K-factor and the shadow fading are applied and a all the data is returned as an '`qd_channel`' object.

Properties

name	Name of the '`qd_builder`' object
scenario	Name of the scenario (text string)
scenpar	The parameter table. See Section 2.4
plpar	Parameters for the path loss. See Section 2.4
simpar	Handle of a '`qd_simulation_parameters`' object. See Section 2.2.1
tx_array	Handles of '`qd_arrayant`' objects for each Tx. See Section 2.2.2
rx_array	Handles of '`qd_arrayant`' objects for each Rx. See Section 2.2.2
rx_track	Handles of '`qd_track`' objects for each Rx. See Section 2.2.3
tx_position	The transmitter position obtained from the corresponding '`qd_layout.tx_position`'
ry_positions	The list of initial positions for which LSPs are generated. This variable is obtained from the properties '`qd_track.initial_position`' and '`qd_layout.rx_position`'
sos	The large-scale parameter SOS generators
gr_sos	The SOS generator for the ground reflection coefficient
path_sos	The SOS generators for the generation of MPCs
xpr_sos	The SOS generators for the generation of the NLOS polarization
pin_sos	The SOS generators for the generation of initial phases
clst_dl_sos	The SOS generators for the generation per-cluster delay offsets
ds	The RMS delay spread in [s] for each receiver position
kf	The Ricean K-Factor [linear scale] for each receiver position
sf	The shadow fading [linear scale] for each receiver position
asD	The azimuth spread of departure in [deg] for each receiver position
asA	The azimuth spread of arrival in [deg] for each receiver position
esD	The elevation spread of departure in [deg] for each receiver position
esA	The elevation spread of arrival in [deg] for each receiver position
xpr	The cross polarization ratio [linear scale] for each receiver position
gr_epsilon_r	The relative permittivity for the ground reflection
NumClusters	The number of clusters
NumSubPaths	The number of sub-paths for each cluster
taus	The initial delays for each path in [s]. Rows correspond to the MTs, columns to the paths.
pow	The normalized initial power (squared average amplitude) for each path. Rows correspond to the MTs, columns to the paths. The sum over all columns must be 1.
AoD	The initial azimuth of departure angles for each path in [rad].
AoA	The initial azimuth of arrival angles for each path in [rad].
EoD	The initial elevation of departure angles for each path in [rad].
EoA	The initial elevation of departure angles for each path in [rad].

gamma	The polarization rotation angle for the lineaar XPR in [rad]. For 3GPP baseline simulations, this property stores the per-path XPR (linear units).
kappa	The phase offset angle for the circular XPR in [rad]. For 3GPP baseline simulations, this property stores the initial random phases. The dimensions correspond to polarization matrix index '[1 3 ; 2 4]', the subpath number and the MT.
pin	The initial phases in [rad] for each sub-path.
subpath_coupling	A random index list for the mutual coupling of subpaths at the Tx and Rx. The dimensions correspond to the subpath index (1-20), the angle (AoD, AoA, EoD, EoA), the path number and the MT.
fbs_pos	The positions of the first-bounce scatterers
lbs_pos	The positions of the last-bounce scatterers
dual_mobility	Indicates if the builder is a dual-mobility builder (-1 = unknown)
no_rx_positions	Number of receiver positions associated to this '`qd_builder`' object Note that each segment in longer tracks is considered a new Rx position.
lsp_vals	The distribution values of the LSPs (extracted from scenpar)
lsp_xcorr_chk	Indicator if cross-correlation matrix is positive definite
lsp_xcorr	The Cross-correlation matrix for the LSPs

Methods

h_builder = **qd_builder** (scenario, check_parfiles)

Calling object	None (constructor)	
Description	Creates a new '`qd_builder`' object.	
Input	scenario	The scenario name for which the parameters should be loaded. A list of supported scenarios can be obtained by calling '`qd_builder.supported_scenarios`'.
	check_parfiles	check_parfiles = 0 / 1 (default: 1) Disables (0) or enables (1) the parsing of shortnames and the validity-check for the config-files. This is useful, if you know that the parameters in the files are valid. In this case, this saves execution time.
Output	h_builder	Handle to the created '`qd_builder`' object.

isdual = **check_dual_mobility**

Calling object	Object array	
Description	Checks the input data of the builder This function checks the input variables "simpar", "tx_array", "rx_array", "tx_track", and "rx_track" of the builder object(s) for conformity. The data representation is adjusted such that all other methods of the builder class can work with the variables without checking them again. The method also checks if dual-mobility processing is required. This information is stored in "qd_builder.dual_mobility" to be accessed by other methods.	
Output	isdual	A logical array indicating for each builder if dual-mobility processing is required (true) or if single-mobility processing is done (false).

gen_parameters (usage)

Calling object	Object array
Description	Generates LSF parameters, SSF parameters and scatterer positions This function generates all parameters that are needed for the channel coefficient generation. The outputs of the method are stored in the class properties. This includes the following steps: 1. Initialize the random generators by calling "`qd_builder.init_sos`". If the random generators are already initialized, use the existing initialization. 2. Generate correlated large scale parameters for each user position. Those parameters are needed by the channel builder to calculate initial SSF parameters for each track or segment which are then evolved into time varying channels. 3. Generates the small-scale-fading parameters for the channel builder. Already existing parameters are overwritten. However, due to the spatial consistency of the model, identical values will be obtained for the same rx positions. Spatial consistency can be disabled by setting "`qd_builder.scenpar.SC_lambda = 0`". 4. Calculates the positions of the scatterers.

Input	usage	Controls the working mode of the method. The allowed options are: • `usage = 0` Clears all exisiting LSF and SSF parameters including the SOS random generators. • `usage = 1` Generates only the LSF parameters. Exisiting LSF parameters will be overwritten and all SSF parameters will be cleared. If the SOS generators are not initialized, they are initialized first. Existing SOS generators in "`qd_builder.sos`" are reused. This leads to identical results when calling the method multiple times. • `usage = 2` Generates the SSF parameters. Exisiting SSF parameters will be overwritten. Existing SOS generators and LSF parameters will be reused. • `usage = 3` Generate the random initial phases, split the SSF parameters into sub-clusters and calculates the scattering cluster positions. This may lead to changes in the departure angles (AoD, EoD). If LSF or SSF parameters are not initialized, they are initialized first. • `usage = 4 (default)` Clears exisiting LSF parameters, SSF parameters and cluster positions and calculates new ones. Existing SOS generators are reused.

angles = **get_angles**

Calling object	Single object	
Description	Calculates the departure- and arrival angles of the LOS path between Tx and Rx	
Output	angles	A matrix containing the four angles: • Azimuth of Departure at the Tx (AoD, row 1) • Azimuth of Arrival at the Rx (AoA, row 2) • Elevation of Departure at the Tx (EoD, row 3) • Elevation of Arrival at the Rx (EoA, row 4) The number of columns corresponds to the number of rx-positions.

h_channel = **get_channels**

Calling object	Object array	
Description	Calculate the channel coefficients	
Output	h_channel	A vector handles to the `qd_channel` objects

dist = **get_distances**

Calling object	Single object	
Description	Calculates the 3D distances between Rx and Tx	
Output	dist	A vector containing the 3D distances between each Rx and the Tx in [m]

h_channel = **get_los_channels** (precision, return_coeff, tx_array_mask)

Calling object	Object array	
Description	Generates channel coefficients for the LOS path only. This function generates static coefficients for the LOS path only. This includes the following properties: • antenna patterns • orientation of the Rx (if provided) • polarization rotation for the LOS path • plane-wave approximation of the phase • path loss • shadow fading No further features of QuaDRiGa are used (i.e. no drifting, spherical waves, time evolution, multipath fading etc.). This function can thus be used to acquire quick previews of the propagation conditions for a given layout.	
Input	precision	The additional input parameter 'precision' can be used to set the numeric precision to 'single', thus reducing the memory requirements for certain computations. Default: double precision.

	return_coeff	Adjusts the format of the output. This is mainly used internally.
		
		`return_coeff = 'coeff';`
		
		If this is set to 'coeff', only the raw channel coefficients are returned, but no QuaDRiGa channel object is created. This may help to reduce memory requirements. In this case, 'h_channel' has the dimensions: [n_rx, n_tx, n_pos]
		
		`return_coeff = 'raw';`
		
		Same as 'coeff' but without applying the distance-dependant phase and the path loss. The rx-antenna is assumed to be dual-polarized with two elements (i.e. the rx interpolation is omitted). This mode is used QuaDRiGa-internally by [`qd_arrayant.combine_pattern`]

	tx_array_mask	Indices for selected transmit antenna elements
Output	h_channel	A handle to the created '`qd_channel`' object. The output contains one coefficient for each position in '`qd_builder.rx_position`'.

map = **get_lsp_map** (xc, yc, zc)

Calling object	Single object	
Description	Calculates the spatial map of the correlated LSPs	
Input	xc	A vector containing the map sample positions in [m] in x-direction
	yc	A vector containing the map sample positions in [m] in y-direction
	zc	A vector containing the map sample positions in [m] in z-direction
Output	map	An array of size [nx, ny, nz, 8] containing the values of the LSPs at the sample positions. The indices of the fourth dimension are: 1. Delay spread [s] 2. K-factor [linear] 3. Shadow fading [linear] 4. Azimuth angle spread of departure [rad] 5. Azimuth angle spread of arrival [rad] 6. Elevation angle spread of departure [rad] 7. Elevation angle spread of arrival [rad] 8. Cross-polarization ratio [linear]

[loss, scale_sf, loss_init, scale_sf_init] = **get_pl** (evaltrack, alt_plpar, txpos)

Calling object	Single object	
Description	Implements various path-loss models This function implements various path-loss models such as defined by 3GPP 36.873 or 38.901. The parameters of the various models are given in the configuration files in the folder '`quadriga_src\config`'. When a builder object is initialized, the parameters appear in the structure '`qd_builder.plpar`'.	
Input	evaltrack	A '`qd_track`' object for which the PL should be calculated. If '`evaltrack`' is not given, then the path loss is calculated for each Rx position. Otherwise the path loss is calculated for the positions provided in '`evaltrack`'.
	alt_plpar	An optional alternative `plpar` which is used instead of '`qd_builder.plpar`'.
	txpos	The corresponding tx positions. This variable can be provided by: 1. A '`qd_track`' object having the same number of snapshots as the '`evaltrack`'; 2. A 3-element vector containing the fixed x,y, and z coordinate of the tx; 3. A [3 x N] matrix containing one tx position for each position on the '`evaltrack`'; 4. The variable is unprovided or empty. In this case, the tx-positions are obtained from the '`qd_builder`' object.
Output	loss	The path loss in [dB]
	scale_sf	In some scenarios, the SF might change with distance between Tx and Rx. Hence, the shadow fading provided by the LSFs generator has to be changed accordingly. The second output parameter "`scale_sf`" can be used for scaling the (logarithmic) SF value.

	loss_init	The path loss in [dB] for the initial position of the "`evaltrack`" object. Returns an empty array if no "`evaltrack`" is given.
	scale_sf_init	"`scale_sf`" for the initial position of the "`evaltrack`" object. Returns an empty array if no "`evaltrack`" is given.

[sf,kf] = **get_sf_profile** (evaltrack, txpos)

Calling object	Single object	
Description	Returns the shadow fading and the K-factor along a track This function returns the shadow fading and the K-factor along the given track. This function is mainly used by the channel builder class to scale the output channel coefficients. The profile is calculated by using the data in the LSF autocorrelation model and interpolating it to the positions in the given track.	
Input	evaltrack	Handle to a '`qd_track`' object for which the SF and KF should be interpolated.
	txpos	The corresponding tx positions. This variable can be provided by: 1. A '`qd_track`' object having the same number of snapshots as the '`evaltrack`'; 2. A 3-element vector containing the fixed x,y, and z coordinate of the tx; 3. A [3 x N] matrix containing one tx position for each position on the '`evaltrack`'; 4. The variable is unprovided or empty. In this case, the tx-positions are obtained from the '`qd_builder`' object.
Output	sf	The shadow fading [linear scale] along the track
	kf	The K-factor [linear scale] along the track

init_sos (force)

Calling object	Object array	
Description	Initializes all SOS random number generators This method initializes all random number generators of the "`qd_builder`" object array. The results are stored in the properties "`sos`", "`gr_sos`", "`path_sos`", "`xpr_sos`", "`pin_sos`", and "`clst_dl_sos`". Once initialized, the "`qd_builder`" object will generate identical parameters and channels. Note that this does not work for 3GPP baseline simulations.	
Input	force	If set to true (1), existing SOS generators will be discarded. By default (0), they will be reused.

h_builder_out = **split_multi_freq**

Calling object	Object array
Description	Split the builder objects for multi-frequency simulations QuaDRiGa allows multi-frequency-simulations by setting multiple carrier frequencies in "qd_simulation_parameters.center_frequency". Correlated SSF for multi-frequency simulations is an additional feature of the 3GPP model (see Section 7.6.5, pp 57 of TR 38.901 V14.1.0). Identical parameters for each Frequency: • LOS / NLOS state must be the same • BS and MT positions are the same (antenna element positions are different!) • Cluster delays and angles for each multipath component are the same • Spatial consistency of the LSPs is identical Differences: • Antenna patterns are different for each frequency • Path-loss is different for each frequency • Path-powers are different for each frequency • Delay- and angular spreads are different • K-Factor is different • XPR of the NLOS components is different

	The method ”split_multi_freq” separates the builder objects so that each builder creates channels for only one frequency. If you call ”split_multi_freq” before any LSF and SSF parameters are generated as it is done the the following code, then LSF parameters (e.g. SF, DS, AS) will be uncorrelated for each frequency. If you call ”gen_lsf_parameters” before ”split_multi_freq”, then all LSF parameters will be fully correlated. However, frequency dependent averages and variances still apply. If you call ”gen_lsf_parameters” and ”gen_ssf_parameters” before ”split_multi_freq”, then SSF will also correlated, i.e. the same paths will be seen at each frequency.	
Output	h_builder_out	An array of ”qd_builder” objects where where each builder only served one frequency.

[scenarios, file_names, file_dir] = **qd_builder.supported_scenarios** (parse_shortnames)

Calling object	None (static method)	
Description	Returns a list of supported scenarios	
Input	parse_shortnames	This optional parameter can disable (0) the shortname parsing. This is significantly faster. By default shortname parsing is enabled (1)
Output	scenarios	A cell array containing the scenario names. Those can be used in ’`qd_track.scenario`’
	file_names	The names of the configuration files for each scenario
	file_dir	The directory where each file was found. You can place configuration file also in your current working directory

visualize_clusters (i_mobile , i_cluster, create_figure)

Calling object	Single object	
Description	Plots the positions of the scattering clusters for a mobile terminal This method plots all scattering clusters for a given mobile terminal. If `i_cluster` is not given, then only the main paths are shown for all MPCs. If `i_cluster` is given, then also the subpaths are shown for the selected cluster. The plot is in 3D coordinates. You can rotate the image using the rotate tool.	
Input	i_mobile	The index of the mobile terminal within the channel builder object
	i_cluster	The index of the scattering cluster. (Optional)
	create_figure	If set to 1 (default), a new figure is created. If set to 0, the last figure is updated

write_conf_file (fn, write_defaults)

Calling object	Single object	
Description	Writes configuration files from qd_builder objects	
Input	fn	String containing the filename
	write_defaults	If set to true, default values are written to the file.

2.2.6 Class “qd_sos”

This class implements the sum-of-sinusoids method for generating spatially correlated random variables. An autocorrelation function (ACF), i.e. a description of the correlation vs. distance, needs to be provided. This function is approximated by a Fourier series. The coefficients of the series can be used to generate spatially correlated random variables.

Properties

name	The name of the object
distribution	Distribution of random variables (’normal’ or ’uniform’)
dist_decorr	Decorrelation distance in [m]
dimensions	Number of dimensions (1, 2 or 3)
no_coefficients	Number of sinusoid coefficients used to approximate the ACF
dist	Vector of sample points for the ACF in [m]
acf	Desired autocorrelation function
sos_freq	Sinusoid coefficients (frequencies)
sos_phase	Phase offsets
sos_amp	Amplitude of the sinusoid coefficients

Methods

h_sos = **qd_sos** (acf_type, distribution, dist_decorr)		
Calling object	None (constructor)	
Description	Creates a new '`qd_sos`' object. Calling `qd_sos` without options creates a new SOS object with default settings. The default ACF is defined by a combination of the Gauss profile (ranging from 0 to the decorrelation distance) and an exponential decaying profile (ranging from the decorrelation distance to a maximum distance of 5 time the decorrelation distance. It is approximated by 300 sinusoids in 3D coordinates. There are 3 optional ACF types that can be selected by `acf_type`. The 3D SOS frequencies are precomputed for 150, 300, 500, and 1000 sinusoids. • Exponential ACF (`Exp150`, `Exp300`, `Exp500`, `Exp1000`) • Gaussian ACF (`Gauss150`, `Gauss300`, `Gauss500`, `Gauss1000`) • Combined Gaussian and Exponential ACF (`Comb150`, `Comb300`, `Comb500`, `Comb1000`) The distributer function can be either ('normal' or 'uniform').	
Input	acf_type	String describing the shape of the autocorrelation function and the number of sinusoids, Default: 'Comb300'
	distribution	Distribution of the random variables ('normal' or 'uniform'), Default: 'normal'
	dist_decorr	Decorrelation distance in [m], Default: 10 m
Output	h_sos	Handle of the created `qd_sos` object

[Ri, Di] = **acf_2d**		
Calling object	Single object	
Description	Interpolates the ACF to a 2D version This method calculates a 2D version of the given ACF (qd_sos.acf). The distance ranges from -2 Dmax to 2 Dmax, where Dmax is the maximum value in qd_sos.dist. Values outside the specified range are set to 0.	
Output	Ri	2D ACF
	Di	Vector of sample points (in x and y direction) for the 2D ACF in [m]

[Ro, Do] = **acf_approx**		
Calling object	Single object	
Description	Generates the approximated ACF from SOS coefficients This method calculates the approximated ACF. The distance ranges from -2 Dmax to 2 Dmax, where Dmax is the maximum value in qd_sos.dist. The format of the output variable (Ro) depends on the number of dimensions. • 1 dimension Ro is a vector containing the values at the sample points of Do. • 2 dimensions Ro is a matrix containing the approximated 2D ACF • 3 dimensions Ro contains 3 matrices, one for the x-y plane, one for the x-z plane and one for the y-z plane.	
Output	Ro	Approximated ACF
	Do	Vector of sample points (in x and y direction) for the ACF in [m]

[Re, De, Re_dual] = **acf_estimate** (no_pos, D)		
Calling object	Single object	
Description	Estimates the 3D and 6D ACF from randomly generated positions This method creates random positions within a cube of 0.4*max(D) m edge length. Then, spatially correlated Normal distributed random numbers are generated for each position using `qd_sos.val`. The distance between each pair of positions is calculated and pairs with similar distance are grouped, i.e. positions with a distance between 0 and 2 meters of each other belonged to group 1, positions with a distance between 2 and 4 meters belonged to group 2, and so on. The Pearson correlation coefficient is calculated for the samples within each group. For the 6D estimation, half the positions are used for the transmitter and half are used for the receiver. Spatially correlated values are generated using the dual-mobility extension. The grouping is done for the receiver positions under the constraint that the transmitter cannot move more than the receiver. For example, if group 3 contains all pairs where the receivers have a distance between 4 and 6 meters, then the transmitters have a distance between 0 and 6 meters.	
Input	no_pos	Number of positions for the ACF estimation.
	D	Distance vector containing the center distances in [m] of the groups.
Output	Re	Estimated 3D ACF
	De	Distance vector containing the actual center distances in [m] of the groups calculated from the random positions.
	Re_dual	Estimated 6D ACF for the dual-mobility extension.

val = **acfi** (dist)		
Calling object	Single object	
Description	Linear interpolation of the ACF	
Input	dist	Vector containing the distances in [m] for which the ACF should be interpolated.
Output	val	Interpolated ACF at the given distances

[mse, mse_core, mse_all, Ro, Ri] = **calc_mse** (T)		
Calling object	Single object	
Description	Calculates the approximation MSE of the ACF This method calculates the mean-square-error (in dB) of the approximation of the given ACF.	
Input	T	Number of test directions, Default: 10
Output	mse	The MSE for the given number of test directions
	mse_core	The 2D MSE for the range 0 to Dmax (covered by the desired ACF)
	mse_all	The 2D MSE for the range 0 to 2*Dmax
	Ro	Approximated ACF from qd_sos.acf_approx
	Ri	Interpolated ACF (from qd_sos.acf_2d in case of 2 dimensions or more)

out = **copy**		
Calling object	Object array	
Description	Creates a copy of the handle class object or array of objects. While the standard copy command creates new physical objects for each element of the object array (in case obj is an array of object handles), copy checks whether there are object handles pointing to the same object and keeps this information.	
Output	out	Copy of the current object or object array

h_sos = **qd_sos.generate** (R, D, N, dim, uniform_smp, debug)

Calling object	None (static method)	
Description	Generates SOS parameters from a given ACF This (static) method approximates any given ACF by sinusoid coefficients. A sampled ACF is provided at the input. Then, the N sinusoid coefficients are iteratively determined until the best match is found.	
Input	R	The desired ACF (having values between 1 and -1). The first value must be 1.
	D	Vector of sample points for the ACF in [m]
	N	Number of sinusoid coefficients used to approximate the ACF
	dim	Number of dimensions (1, 2 or 3)
	uniform_smp	If set to 1, sample points for 2 or more dimensions are spaced equally. If set to 0, sample points are chosen randomly. Default: 0 (random)
	T	Number of test directions for the MSE calculation, Default: 10
	random_init	If set to 1, the SOS frequencies are randomly initialized. If set to 0, the frequencies are initialized by zeros. Default: 1 (random)
	show_progress	If set to 1, an animation plot of the progress is shown. Default: 0
Output	h_sos	Handle of the created `qd_sos` object

init (use_same)

Calling object	Object array	
Description	Initializes the random phases	
Input	use_same	If set to 1 (default), identical phases are used for the dual-mobility option, assuming that devices are on the same radio-map. If set to 0, different values are used.

h_sos = **qd_sos.load** (filename)

Calling object	None (static method)	
Description	Loads coefficients from mat file Sinusoid coefficients can be stored in a mat-file by calling `qd_sos.save`. This (static) method loads them again. In this way, it is possible to precompute the sinusoid coefficients and save some significant time when initializing the method. It is possible to adjust the decorrelation distance of a precomputed function without the need to perform the calculations again.	
Input	filename	Path or filename to the coefficient file
Output	h_sos	Handle of the loaded `qd_sos` object

S = **map** (xc, yc, zc)

Calling object	Single object	
Description	Generates random variables at the given coordinates in x,y and z direction This method generates a multi-dimensional array (of up to 3 dimensions) of spatially correlated random variables.	
Input	xc	A vector containing the map sample positions in [m] in x-direction
	yc	A vector containing the map sample positions in [m] in y-direction
	zc	A vector containing the map sample positions in [m] in z-direction
Output	S	Array of spatially correlated random variables

[s, h_sos] = **qd_sos.rand** (dist_decorr, ca, cb)

Calling object	None (static method)	
Description	Generates spatially correlated uniformly distributed random numbers	
Input	dist_decorr	Vector of decorrelation distances [1 x M] or [M x 1]
	ca	Coordinates for the first mobile device in [m] given as [3 x N] matrix. The rows correspond to the x,y and z coordinate.
	cb	Coordinates for the corresponding second mobile device in [m] given as [3 x N] matrix. The rows correspond to the x,y and z coordinate. This variable must either be empty or have the same size as "ca".
	acf_type	String describing the shape of the autocorrelation function and the number of sinusoids, Default: 'Comb300'
Output	s	Random spatially correlated numbers [M x N]
	h_sos	A handle to the used `qd_sos` object

[s, h_sos] = **qd_sos.randi** (dist_decorr, ca, imax, cb)		
Calling object	None (static method)	
Description	Generates spatially correlated random integers between 1 and imax	
Input	dist_decorr	Vector of decorrelation distances [1 x M] or [M x 1]
	ca	Coordinates for the first mobile device in [m] given as [3 x N] matrix. The rows correspond to the x,y and z coordinate.
	imax	The maximum value [1 x 1]
	cb	Coordinates for the corresponding second mobile device in [m] given as [3 x N] matrix. The rows correspond to the x,y and z coordinate. This variable must either be empty or have the same size as "ca".
	acf_type	String describing the shape of the autocorrelation function and the number of sinusoids, Default: 'Comb300'
Output	s	Random spatially correlated numbers [M x N]
	h_sos	A handle to the used `qd_sos` object

[s, h_sos] = **qd_sos.randn** (dist_decorr, ca, cb)		
Calling object	None (static method)	
Description	Generates spatially correlated normal distributed random numbers	
Input	dist_decorr	Vector of decorrelation distances [1 x M] or [M x 1]
	ca	Coordinates for the first mobile device in [m] given as [3 x N] matrix. The rows correspond to the x,y and z coordinate.
	cb	Coordinates for the corresponding second mobile device in [m] given as [3 x N] matrix. The rows correspond to the x,y and z coordinate. This variable must either be empty or have the same size as "ca".
	acf_type	String describing the shape of the autocorrelation function and the number of sinusoids, Default: 'Comb300'
Output	s	Random spatially correlated numbers [M x N]
	h_sos	A handle to the used `qd_sos` object

save (filename)		
Calling object	Single object	
Description	Saves the coefficients to a mat-file Sinusoid coefficients can be stored in a mat-file. In this way, it is possible to precompute the sinusoid coefficients and save significant time when initializing the method. It is possible to adjust the decorrelation distance of a precomputed function without the need to perform the calculations again.	
Input	filename	Path or filename to the coefficient file

h_sos = **qd_sos.set_acf** (acf_type, distribution, dist_decorr)		
Calling object	None (static method)	
Description	Creates a new '`qd_sos`' object. There are 3 ACF types that can be selected by `acf_type`. The 3D SOS frequencies are precomputed for 150, 300, 500, and 1000 sinusoids. • Exponential ACF (`Exp150`, `Exp300`, `Exp500`, `Exp1000`) • Gaussian ACF (`Gauss150`, `Gauss300`, `Gauss500`, `Gauss1000`) • Combined Gaussian and Exponential ACF (`Comb150`, `Comb300`, `Comb500`, `Comb1000`) The distributer function can be either ('normal' or 'uniform').	
Input	acf_type	String describing the shape of the autocorrelation function and the number of sinusoids, Default: 'Comb300'
	distribution	Distribution of the random variables ('normal' or 'uniform'), Default: 'normal'
	dist_decorr	Decorrelation distance in [m], Default: 10 m
Output	h_sos	Handle of the created `qd_sos` object

s = **val** (ca, cb)		
Calling object	Object array	
Description	Returns correlated values at given coordinates This method generates spatially correlated random variables at the given coordinates. The function allows two inputs for the coordinates ("ca" and "cb"). This can be used for dual-mobility scenarios where both ends of the link are moving. In this case, "ca" contains the coordinates for the first mobile device and "cb" contains the coordinates for the second mobile device. For each point in "ca" there must be one point in "cb". For single-mobility scenarios, where only one side is mobile, you may only provide the input for "ca".	
Input	ca	Coordinates for the first mobile device in [m] given as [3 x N] matrix. The rows correspond to the x,y and z coordinate.
	cb	Coordinates for the corresponding second mobile device in [m] given as [3 x N] matrix. The rows correspond to the x,y and z coordinate. This variable must either be empty or have the same size as "ca".
Output	s	Vector ([M x N] elements) of spatially correlated random variables. If the method was called on an object arrays, the rows contain the outputs of the individual SOS objects.

2.2.7 Class "qd_channel"

Objects of this class are the output of the channel model. They are created by '`qd_builder`' objects. By default, channel coefficients are provided in the time domain, as a list of delays and complex-valued amplitudes. However, this class also implements certain methods to postprocess the channel data. Those include:

- Transformation into frequency domain
- Interpolation in time domain (to change the terminal speed and sampling rate)
- Combining channel traces into longer segments (including birth and death of clusters)

Properties

name	Name of the '`channel`' object. This string is a unique identifier of the '`qd_channel`' object. The '`qd_builder`' creates one channel object for each MT, each Tx and each segment. They are further grouped by scenarios (propagation environments). The string consists of four parts separated by an underscore '_'. Those are: • The scenario name from '`qd_track.scenario`' • The transmitter name from '`qd_layout.tx_name`' • The receiver name from '`qd_layout.rx_name`' • The segment number After '`qd_channel.merge`' has been called, the name string consists of: • The transmitter name from '`qd_layout.tx_name`' • The receiver name from '`qd_layout.rx_name`'
version	Version number of the QuaDRiGa release that was used to create the '`qd_channel`' object.
center_frequency	Center frequency in [Hz].
coeff	The complex-valued channel coefficients for each path. The indices of the 4-D tensor are: [no_rxant , no_txant , no_path , no_snapshot]
delay	The delays for each path. There are two different options. If the delays are identical on the MIMO links, i.e. '`individual_delays = 0`', then '`delay`' is a 2-D matrix with dimensions [no_path , no_napshot]. If the delays are different on the MIMO links, then '`delay`' is a 4-D tensor with dimensions [no_rxant , no_txant , no_path , no_snapshot].
par	Field to store custom variables.
initial_position	The snapshot number for which the initial LSPs have been generated. Normally, this is the first snapshot. However, if the user trajectory consists of more than one segment, then '`initial_position`' points to the snapshot number where the current segment starts. For example: If '`initial_position`' is 100, then snapshots 1-99 are overlapping with the previous segment.

tx_position rx_position	Position of each Tx in global cartesian coordinates using units of [m]. The receiver position global cartesian coordinates using units of [m] for each snapshot.
no_rxant no_txant no_path no_snap	Number of receive elements (read only) Number of transmit elements (read only) Number of paths (read only) Number of snapshots (read only)
individual_delays	Indicates if the path delays are identical on each MIMO link (0) or if each link has a different path delay (1).

Methods

h_channel = **qd_channel** (Ccoeff, Cdelay, Cinitial_position)

Calling object	None (constructor)	
Description	Creates a new 'qd_channel' object.	
Input	Ccoeff	The complex-valued channel coefficients for each path. The indices of the 4-D tensor are: [no_rxant , no_txant , no_path , no_snapshot]
	Cdelay	The delays for each path. There are two different options: a 2-D matrix with dimensions [no_path , no_snapshot] or a 4-D tensor with dimensions [no_rxant , no_txant , no_path , no_snapshot].
	Cinitial_position	The snapshot number for which the initial LSPs have been generated
Output	h_channel	Handle to the created 'qd_channel' object

h_channel_comb = **combine_tx_rx**

Calling object	Object array	
Description	Combines channels from a `qd_channel` array into a single object This method operates on an array of `qd_channel` objects containing channels from several BSs to several MTs. It then merges these channels into a single `qd_channel` object. For example, if you use QuaDRiGa to generate channels from two single-antenna BSs to one single-antenna MT, you get an two-element array of `qd_channel` objects. The first object contains the SISO channel from BS1 to the MT, the second object contains the SISO channel from BS2 to the MT. However, for some evaluations, it would be preferable to have a MISO channel where the two BSs operate as a distributed-antenna system. This is done by this method. If you call "`combine_tx_rx`" on the `qd_channel` array, you get a MISO channel object with two transmit antennas and one receive antenna as output. Note that for mobile receivers, all individual channel objects must have the same number of snapshots.	
Output	h_channel_comb	Single `qd_channel` object containing the channel coefficients of the combined channel.

out = **copy**

Calling object	Object array	
Description	Creates a copy of the handle class object or array of objects. While the standard copy command creates new physical objects for each element of the object array (in case obj is an array of object handles), copy checks whether there are object handles pointing to the same object and keeps this information.	
Output	out	Copy of the current object or object array

freq_response = **fr** (bandwidth, carriers, i_snapshot)		
Calling object	Single object	
Description	Transforms the channel into frequency domain and returns the frequency response	
Input	bandwidth	The baseband bandwidth in [Hz]
	carriers	The carrier positions. There are two options: 1. Specify the total number of carriers. In this case, '`carriers`' a scalar natural number > 0. The carriers are then equally spaced over the bandwidth. 2. Specify the pilot positions. In this case, '`carriers`' is a vector of carrier positions. The carrier positions are given relative to the bandwidth where '0' is the begin of the spectrum and '1' is the end. For example, if a 5 MHz channel should be sampled at 0, 2.5 and 5 MHz, then '`carriers`' must be set to [0, 0.5, 1].
	i_snapshot	The snapshot indices for which the frequency response should be calculated. By default, i.e. if '`i_snapshot`' is not given, all snapshots are processed.
Output	freq_response	The complex-valued channel coefficients for each carrier in frequency domain. The indices of the 4-D tensor are: [Rx-Antenna , Tx-Antenna , Carrier-Index , Snapshot]

c = **interpolate** (dist, algorithm)		
Calling object	Single object	
Description	Interpolates the channel coefficients and delays. The channel builder creates one snapshot for each position that is listed in the track object. When the channel sampling theorem is not violated (i.e. the sample density is ≥ 2), then the channel can be interpolated to any other position on the track. This can be used e.g. to emulate arbitrary movements along the track. For more information see '`qd_track.movement_profile`', '`qd_track.interpolate_movement`', or the tutorial “Applying Varying Speeds (Channel Interpolation)”.	
Input	dist	A vector containing distance values on the track. The distance is measured in [m] relative to the beginning of the track. Alternatively, “dist” can be given as a 3-D tensor with dimensions [Rx-Antenna , Tx-Antenna , Snapshot]. In this case, interpolation os done for each antenna element separately.
	algorithm	Selects the interpolation algorithm. The default is linear interpolation. Optional are: • linear - Linear interpolation (optimized for speed) • cubic - Cubic spline interpolation of the channel coefficients and piecewise cubic hermite polynomial interpolation for the delays
Output	c	Handle to the '`qd_channel`' object containing the interpolated coefficients and delays for each entry in 'dist'.

c = **merge** (overlap, optimize, verbose)	
Calling object	Object array
Description	Combines channel segments into a continuous time evolution channel. The channel merger implements the continuous time evolution with smooth transitions between segments. Each segment of a track is split in two parts: an overlapping area with the previous segment and an exclusive part with no overlapping. Each segment is generated independently by the channel builder. However, the distance dependent autocorrelation of the large scale parameters was considered when the parameters were drawn from the corresponding statistics. Transition from segment to segment is carried out by replacing taps of the previous segment by the taps of the current segment, one by one. The modeling of the birth/death process is done as published in the documentation of the WIM2 channel model. The route between adjacent channel segments is split into sub-intervals equal to the minimum number of taps in both overlapping segments. During each sub-interval the power of an old tap ramps down and a new tap ramps up. Power ramps are modeled by a modified sinus function to allow smooth transitions.

	Taps from the old and new segments are coupled based on their power. If the number of clusters is different in the channel segments, the weakest clusters are ramped up or down without a counterpart from the new/old segment. The merging is only done for the NLOS components since the LOS component has a deterministic behavior. The LOS component is thus just scaled in power.	
Input	overlap	The length of the overlapping part relative to the segment length. It can have values in between 0 (no overlap) and 1 (ramp along the entire segment). A value of 0 disables the merging process and the channel segments are simply concatenated. A value of 1 constantly merges the channels. The default setting is 0.5.
	verbose	Enables (1, default) or disables (0) the progress bar.
Output	c	An array of handles to the '`qd_channel`' objects containing the merged coefficients and delays.

h_channel_quant = **quantize_delays** (tap_spacing, verbose)

Calling object	Single object	
Description	Fixes the path delays to a grid of delay bins For some applications, e.g. channel emulation, it is not possible to achieve an infinite delay accuracy. However, when the delays are rounded to a fixed grid of delay-bins (also refereed to as "taps"), the time-evolving channel is no longer smooth. When a delay "jumps" from one delay-bin to the next, e.g. when a MT is moving away from the BS, the phases in the frequency domain representation of the channel will suddenly change as well. Multi-carrier communications systems with closed-loop channel adaption (e.g. OFDM, WiFi, LTE, etc.) will exhibit poor performance in this case. This method corrects this problem by approximating the "real" delay value by two delays at a fixed spacing. For example: when we assume that the required tap spacing is 5 ns (which corresponds to a 200 MHz sample-rate) and the distance between BS and MT is 10 m, the delay of the LOS path would be 33.4 ns. However, the fixed tap spacing only allows values of 30 or 35 ns. This method approximates the LOS delay by two taps (one at 30 and one at 35 ns) and linear interpolation of the path power. Hence, in the frequency domain, the transition from one tap to the next is smooth. But note: this only works when the bandwidth of the communication system is significantly less than the sample-rate.	
Input	tap_spacing	The spacing of the delay-bin in [seconds]. Default: 5 ns
	verbose	Enables (1, default) or disables (0) a progress bar.
Output	h_channel_quant	A `qd_channel` object containing the approximated delays and channel coefficients

chan_out = **split_snap** (varargin)

Calling object	Single object	
Description	Splits channel objects based on snapshot indices. This function can be used to split a channel object into sub-objects based on a list of snapshots. For example, this can be used to separate channels into LOS and NLOS parts. To split the channels, the following command can be used: cs = c.split(1:100, 101:2:200); This splits the channel object "c" into two sub-channels, the first containing the snapshots 1 to 100, and the second containing the snapshots 101 to 199 (at half resolution). Notes: • Inputs must be scalar channel objects. • If there is evaluation data in the `par` field, it will be split as well. This requires the field `par.cluster_ind` which determines the small-scale-fading averaging intervals. • A running index (in the format "p001", "p002", etc.) is added to the channel name, so that the sub-channels can be identified.	
Input	varargin	A list of snapshot indices. The number of inputs determines the number of output channels.
Output	chan_out	An array of handles to the split '`qd_channel`' objects

chan_out = **split_tx** (varargin)		
Calling object	Object array	
Description	Splits channel arrays based on transmit antenna indices. This function can be used to split large transmit array antennas into smaller arrays. For example, this can be used to calculate the channels for individual sectors at a BS. Example: A channel array has channels from three base stations (BSs). The first and second BS have two sectors, each with two antennas. However, the sector antennas are merged into one array. The third BS has only one sector. To split the channels into five sectors, the following command can be used: cs = c.split({1:2,3:4}, {1:2,3:4}, {1:2}); Notes: • The method parses the name-string of the channel objects `channel.name` in order to determine the Tx-Rx relationship. There are two allowed formats: (a) ”tx_rx” and (b) ”scenario_tx_rx” • The order of the inputs must match the transmitters in alphabetical order, i.e. the first input corresponds to ”Tx01”, the second to ”Tx02” and so on. This is independent of the order in ”layout.tx_name”, which might have a different order. • If only one cell is given as input, but there are several Txs in the channel array, the same sectorization is applied to each one of them. • Outputs are sorted alphabetically according to ”tx_rx” (scenario names are ignored) • If the input array is shaped as [Rx, Tx], the output will be shaped as [Rx, Tx * Sec]	
Input	varargin	A list of cell-arrays containing the transmit antenna indices.
Output	chan_out	An array of handles to the split ’`qd_channel`’ objects

2.2.8 QuaDRiGa package functions

The QuaDRiGa package functions are independent functions that are use by several classes of the QuaDRiGa channel model. They can be fount in the ”`+qf`” folder and are called by ”qf.function_name”.

Functions

[Sh , bins , Sc , mu , sig] = **qf.acdf** (data , bins , dim , cdim)		
Calling object	None (static method)	
Description	Calculate the cumulative distribution function of a given data set This function calculates the empirical cumulative distribution function from the given `data`. It is possible to analyze multi-dimensional data-sets and average the results. For example, you may collect 10,000 data samples of an experiment and repeat the experiment 5 times. Hence, your `data` variable may have the size [5,10000]. Then, you want to calculate 5 independent CDFs, one for each experiment run, and evaluate how much the results differ for each run. Calling `[ Sh , bins, Sc , mu , sig ] = qf.acdf( data, [], 2,1 );` will produce the desired results. The input parameter `dim = 2` describes which dimension of the data set contains the samples, `cdim = 1` describes on which dimension the repetitions are stored. The output variable `Sh` contains the individual probabilities (y-axis), `bins` contains the sample values (x-axis), `Sc` the average probabilities where the averaging was done over the quantiles.	
Input	data	The data samples can be either given as a multi-dimensional array of size [nA, nB, nC ...] or as a cell array. The latter can be useful if there are different numbers of results per experimental run.
	bins	The center values of each bin on the x-axis of the (non-cumulative) distribution function. By default, 201 bins are equally spaced over the data range.
	dim	The dimension on which the analysis is done, Default: 1
	cdim	The dimension for which the resulting CDFs are averaged, Default: 2
Output	Sh	The individual CDFs. Default size: [no_bins, nB, nC, ...]

	bins	The center values of each bin on the x-axis of the (non-cumulative) distribution function.
	Sc	The averaged CDFs; Default size: [no_bins, nC, ...]
	mu	The average 0.1 ... 0.9 quantiles; Default size: [9, nC, ...]
	sig	The standard deviation of the 0.1 ... 0.9 quantiles; Default size: [9, nC, ...]

[az, el, J, pow, accuracy, resolved] = **qf.calc_angles** (cf, h_qd_arrayant, no_subpath, range, verbose)

Calling object	None (static method)	
Description	Estimates the arrival angles from the channel coefficients This function estimates the arrival angles from a given coefficient matrix `cf`. This is done by comparing the coefficients with the antenna response of the given antenna object `h_qd_arrayant` and determining the most likely arrival direction in azimuth and elevation relative to the local antenna coordinate system. Note that the direction finding results depend on the spatial resolution capabilities of the antenna. If the array antenna has no spatial resolution (e.g. a single dipole), then the returned values will be incorrect.	
Input	cf	The channel coefficients; size [n_rx, n_tx, n_coeff]
	h_qd_arrayant	The receive antenna object. The number of elements must match [n_rx]
	no_subpath	The maximum number of sub-paths per path, default: 1
	range	The allowed angle range in degree [-el +el -az +az], Default: [-90 90 -180 180]
	verbose	Show the estimation progress. 0 = Disable, 1 = Show progress bar, 2 = Show spatial signature
Output	az	The azimuth angles in [rad], dimension [n_coeff, no_subpath]
	el	The elevation angles in [rad], dimension [n_coeff, no_subpath]
	J	The Jones vectors of the path, dimension [2, n_tx, n_coeff, no_subpath]
	pow	The power of the path without antenna pattern [n_tx, n_coeff, no_subpath]
	accuracy	The estimation MSE [n_coeff, 1]
	resolved	The resolved spatial power [n_coeff, 1]

[as, mean_angle] = **qf.calc_angular_spreads** (ang, pow, wrap_angles)

Calling object	None (static method)	
Description	Calculates the angular spread in [rad] This function calculates the RMS angular spread from a given set of angles. It is used by the "qd_builder" to map the path powers to angles.	
Input	ang	A vector of angles in [rad]. Dimensions: [n_ang x n_path]
	pow	A vector of path powers in [W]. Dimensions: [n_ang x n_path] or [1 x n_path]
	wrap_angles	A logical variable. If set to 1, angles will be wrapped around +/- pi. If set to 0, no wrapping is applied. Default: 1 (with wrapping)
Output	as	The RMS angular spread in [rad] for each angle vector. Dimensions: [n_ang x 1]
	mean_angle	The mean angles in [rad]. Dimensions: [n_ang x 1]

[R, phiL, thetaL, gamma] = **qf.calc_ant_rotation** (zrot, yrot, xrot, phi, theta)

Calling object	None (static method)	
Description	Calculates the polarimetric antenna rotation in geographic coordinates This function implements the antenna rotation, including the polarization. For each rotation angle "zrot", "yrot", and "xrot", a 3D rotation matrix (Rz, Ry, Ry) is calculated (assuming a right-handed Cartesian coordinate system). The matrices are combined in the order "Rz * Ry * Rx", i.e. the x-rotation is applied first and the z-rotation is applied last. The input angles "phi" (azimuth) and "theta" (elevation) are relative to the global coordinate system. They are converted into the local antenna coordinates "phiL" and "thetaL" which takes the antenna orientation into account. The angle "gamma" is the polarization rotation angle. The rotation angles "zrot", "yrot", and "xrot" can be empty, scalar, or match the size of "phi and "theta". This function is used by the classes "qd_arrayant", "qd_layout", and "qd_builder". Note: In the "qd_track" objects, the property "height_direction" is defined as a NEGATIVE rotation around the y-axis (right-hand-rule), and the "ground_direction" is a positive rotation around the z-axis. Hence, you need to use the negative "height_direction" as "yrot" to get the correct antenna rotation!	
Input	zrot	Rotation angle around z-axis in [rad], Default = 0
	yrot	Rotation angle around y-axis in [rad], Default = 0

	xrot	Rotation angle around x-axis in [rad], Default = 0
	phi	Azimuth input angles in [rad], Default = 0
	theta	Elevation input angles in [rad], Default = 0
Output	R	Antenna rotation matrix (Rz * Ry * Rx)
	phiL	Azimuth angles for pattern interpolation in [rad]
	thetaL	Elevation angles for pattern interpolation in [rad]
	gamma	The polarization rotation angle in [rad]

[ds, mean_delay] = **qf.calc_delay_spread** (taus, pow)

Calling object	None (static method)	
Description	Calculates the delay spread in [s] This function calculates the RMS delay spread from a given set of delays and powers. It is used by the "qd_builder" to map the path powers to delays.	
Input	ang	A vector of deays [s]. Dimensions: [n_taus x n_path]
	pow	A vector of path powers in [W]. Dimensions: [n_taus x n_path]
Output	as	The RMS delay spread for each delay vector. Dimensions: [n_taus x 1]
	mean_angle	The mean delay in [s]. Dimensions: [n_taus x 1]

iseq = **qf.eqo** (obj, obj_array)

Calling object	None (static method)	
Description	Determines if object handles are equal Octave 4.0 does not implement the "eq" function and is very sensitive to incorrect indexing of object arrays. This function provides the required functionality for QuaDRiGa. For this to work, the corresponding classes must have a writable property "OctEq" which may be hidden.	
Input	obj	A single object handle.
	obj_array	A (multi-dimensional) array of object handles with up to four dimensions.
Output	iseq	A boolean variable having the same size as "obj_array". True (1) values indicate that the corresponding handles in "obj_array" point to the same object as the "obj"-handle does.

zi = **qf.interp** (x, y, z, xc, yc)

Calling object	None (static method)	
Description	2D linear interpolation optimized for performace This function implements a 2D linear interpolation which is highly optimized for fast execution. All calculations are done in single-precision floating point (30% faster than double precision, but less accurate), and multiple data sets can be interpolated simultaneously. One-dimensional linear interpolation can be done by using `zi = interp( x, 0, z, xc )`	
Input	x	Vector of x sample points; size [1, nx] or [nx, 1]
	y	Vector of y sample points; size [1, ny] or [ny, 1]
	z	The input data matrix; size [ny, nx, ne]; the 3rd dimension allows for interpolations of multiple data-sets; for one-dimensional interpolation, the size must be [1, nx, ne]
	xc	Vector of x sample points after interpolation; size [1, nxi] or [nxi , 1]
	xc	Vector of y sample points after interpolation; size [1, nyi] or [nyi, 1]
Output	zi	The interpolated data; size [nyi, nxi, ne]

[mu, epsilon, gamma, Ri, sigma, kappa, delta] = **qf.log2dfit** (vi, ai, bi, round_digits, show_plot, a_linear, b_linear, a_name, b_name, v_name)

Calling object	None (static method)

Description	Logarithmic curve fitting with two variables
	
	This function fits the input data "`vi`" sampled at points "`ai`" and "`bi`" to the following model in a least square sense:
	
	`V = R + X * S`
	`R = mu + epsilon * log10( a ) + gamma * log10( b )`
	`S = sigma + kappa * log10( a ) + delta * log10( b )`
	
	`V` is assumed to be a random, normal-distributed variable with a mean value `R` and standard deviation `S`. The reference value `R` depends on the variables `a` and `b`. `X` is a normal-distributed random variable with zero-mean and unit-variance. The scaling of the STD of `V` is done by `S`, which also depends on the variables `a` and `b`.	

Input	vi	Vector of input data values
	ai	Sample points for variable `a` (linear). If `ai` is empty, 1D fitting is done for `bi` only.
	bi	Sample points for variable `b` (linear). If `bi` is empty, 1D fitting is done for `ai` only. If `ai` and `bi` are empty, only `mu` and `sigma` are returned.
	round_digits	Rounds the output to this number of decimal digits after the coma.
	show_plot	Shows a plot with the results and outputs the fitted values to the console. The plot contains the average and the 1-sigma interval above and below the average as well as the data points. The options are: (0) disables the plot; (1, default) shows the plot and the text for `avg` and `std`; (2) shows the plot and the text for `avg` only; (3) shows the plot and the text for `std` only; (4) shows the plot only
	a_linear	If set to true, fitting is done for linear values of `ai` instead of `log10( ai )`.
	b_linear	If set to true, fitting is done for linear values of `bi` instead of `log10( bi )`.
	a_name	Alternative variable name for `a` in the figure title and console text
	b_name	Alternative variable name for `a` in the figure title and console text
	v_name	Alternative name for the z-axis in the figure
Output	mu	Reference value `R` at `a = 1` and `b = 1`
	epsilon	Scaling of the reference vale `R` with `a` [`R/log10(a)`] or [`R/a`]
	gamma	Scaling of the reference value `R` with `b` [`R/log10(b)`] or [`R/b`]
	Ri	The reference value `R` for the input sample points `ai` and `bi`
	sigma	The STD of `V` at `a = 1` and `b = 1`
	kappa	Scaling of the STD `S` with `a` [`S/log10(a)`] or [`S/a`]; If `kappa` is not requested as an output variable, no STD scaling is assumed.
	delta	Scaling of the STD `S` with `b` [`S/log10(b)`] or [`S/b`]; If `delta` is not requested as an output variable, STD scaling is only done for `a`.

str = **qf.mat2str** (dat, name, line, format)

Calling object	None (static method)	
Description	Converts numeric data into M-code This function converts numeric data into M-code that can be used in MATLAB/Octave to be executed on the command line or embedded into scripts and functions.	
Input	dat	A multi-dimensional array containing the real-valued data. The maximum number of dimension is 3.
	name	The variable name (string) to be written to the output.
	line	Number of characters per line. Default: 100
	format	Format of the output fields, specified as a string. See "num2str" for reference.
Output	str	Formatted output string.

[theta, phi, B, d_phi] = **qf.pack_sphere** (N)

Calling object	None (static method)	
Description	Creates equally distributed points on the unit sphere This function equally distributes points on a unit sphere. The actual number of placed points (M) might be smaller then N due to rounding offsets.	
Input	N	The number of points to place.
Output	theta	Vector of elevation angles in [rad] having values from -pi/2 to pi/2; size [M x 1]
	phi	Vector of azimuth angles in [rad] having values from -pi to pi; size [M x 1]
	B	Positions of the points on Cartesian coordinates; size [3 x M]

[i1,i2,i3,i4] = **qf.qind2sub** (sic, ndx)

Calling object	None (static method)	
Description	Calculates subscripts from linear indices. The `qind2sub` command determines the equivalent subscript values corresponding to a single index into an array. The output is identical to the MATLAB / Octave ind2sub command. However, the `qind2sub` command offers slightly better performance.	
Input	sic	A vector describing the layout of the object. For two-dimensional objects, siz(1) is the number of rows and siz(2) is the number of columns.
	ndx	A vector containing the linear indices of the array.
Output	i1	Row-index
	i2	Column-index
	i3	Page-index
	i4	Index of the 4-th dimension

out = **qf.reshapeo** (obj, shape)

Calling object	None (static method)	
Description	Reshapes the input handle object array to an output object array Octave 4.0 does not implement the "reshape" function and is very sensitive to incorrect indexing of object arrays. This function provides the required functionality for QuaDRiGa.	
Input	obj	Object array with up to 4 dimensions.
	shape	A vector describing the desired layout of the object array. For two-dimensional array, shape(1) is the number of rows and shape(2) is the number of columns.
Output	out	Reshaped object array with up to 4 dimensions.

[phiI, thetaI, pI] = **qf.slerp** (x, phi, theta, xi)

Calling object	None (static method)	
Description	Spherical linear interpolation optimized for performace Slerp is shorthand for spherical linear interpolation and refers to constant-speed motion along a unit-radius great circle arc. This function is needed when interpolating angles in spheric or circular coordinates, e.g. when interpolating phase information (`qd_channel.interpolate`) or orientations (`qd_track.interpolate`). Circular linear interpolation can be done by using `phiI = slerp( x, phi, 0, xi )`	
Input	x	Vector of x sample points; size [1, nx] or [nx, 1]
	phi	Vector of phi angles in [rad]; values from -pi to pi; size [nx, ne]; the 2nd dimension allows for interpolations of multiple data-sets
	theta	Vector of theta angles in [rad]; values from -pi/2 (down) to pi/2 (up); size [nx, ne]; the 2nd dimension allows for interpolations of multiple data-sets
	xc	Vector of x sample points after interpolation; size [1, nxi] or [nxi , 1]
Output	phiI	Vector of interpolated phi angles in [rad]; values from -pi to pi; size [nxi, ne]
	thetaI	Vector of interpolated theta angles in [rad]; values from -pi/2 (down) to pi/2 (up); size [nxi, ne]
	pI	Cartesian coordinates of the interpolated points on the unit-sphere; size [3, nxi, ne]

c = **qf.xcorrcoeff** (a,b)

Calling object	None (static method)	
Description	Calculates the correlation coefficient of two vectors	
Input	a	Data vector of size [1 x N]
	b	Data vector of size [T x N]
Output	c	Correlation coefficient between the values in a and b, size [1 x T]

2.3 Data Flow

The data flow of the QuaDRiGa channel model is depicted in Fig. 5. This figure shows how each of the processing steps, which are described in detail in the following sections, are linked together. The lines show, which parameters are exchanged and how often they are updated. Black lines are for parameters that are either provided by the model users or which are given in the parameter table. Those values are constant. Blue values are updated once per segment and red values are updated once per snapshot.

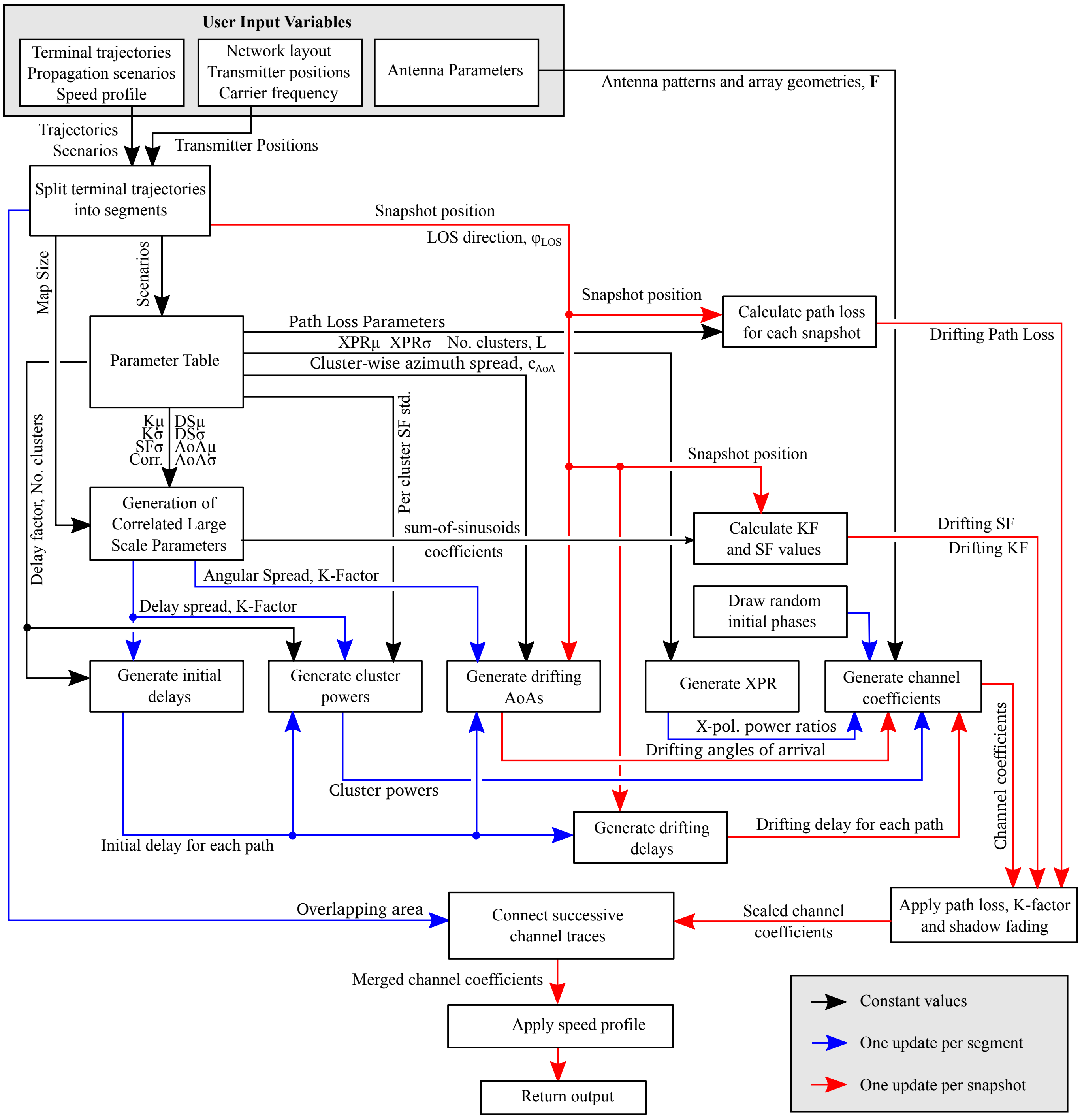

User Input Variables
Terminal trajectories Propagation scenarios Speed profile
Network layout Transmitter positions Carrier frequency
Antenna Parameters
Antenna patterns and array geometries, F
Trajectories Scenarios
Transmitter Positions
Split terminal trajectories into segments
Snapshot position
LOS direction, φLOS
Map Size
Scenarios
Snapshot position
Calculate path loss for each snapshot
Drifting Path Loss
Path Loss Parameters
XPRμ XPRσ No. clusters, L
Cluster-wise azimuth spread, cAoA
Parameter Table
Kμ Kσ SFσ Corr. DSμ DSσ AoAμ AoAσ
Per cluster SF std.
Delay factor, No. clusters
Snapshot position
Generation of Correlated Large Scale Parameters
sum-of-sinusoids coefficients
Calculate KF and SF values
Drifting SF
Drifting KF
Angular Spread, K-Factor
Delay spread, K-Factor
Draw random initial phases
Generate initial delays
Generate cluster powers
Generate drifting AoAs
Generate XPR
Generate channel coefficients
X-pol. power ratios
Drifting angles of arrival
Cluster powers
Channel coefficients
Initial delay for each path
Generate drifting delays
Drifting delay for each path
Overlapping area
Connect successive channel traces
Scaled channel coefficients
Apply path loss, K-factor and shadow fading
Merged channel coefficients
Apply speed profile
Return output
Constant values
One update per segment
One update per snapshot

Figure 5: QuaDRiGa Data Flow

2.4 Scenario Specific Parameters

The large-scale parameters (LSPs) are defined by the parameter files which can be found in the folder '`config`' of the QuaDRiGa source ('`quadriga_src`') folder. The method '`qd_layout.set_scenario`' can be used to assign parameters automatically to receivers. A scenario may include several files describing propagation conditions for line of sight (LOS), non-line of sight (NLOS), and outdoor-to-indoor (O2I). The following table describes the parameter groups and their applicability.

Table 29: Parameter sets provided together with the standard software

Generic parameter sets for specific QuaDRiGa functions	
`LOSonly`	One LOS path only, no shadow fading, no path loss
`Freespace`	One LOS path only, no shadow fading, freespace loss
`TwoRayGR`	Two-Ray Ground Reflection model (2 paths), no shadow fading, freespace loss
`Null`	A parameter set that disables the channel (e.g. for satellites below the horizon)

3GPP 3D model (3GPP TR 36.873) [9]	
`3GPP_3D_UMa` - 3GPP 3D Urban Macro-Cell	

`3GPP_3D_UMa_LOS`
`3GPP_3D_UMa_LOS_O2I`
`3GPP_3D_UMa_NLOS`
`3GPP_3D_UMa_NLOS_O2I`
	For typical terrestrial base stations deployed above rooftop in densely populated urban areas. Parameters cover LOS, NLOS and O2I. The maximum cell radius is about 1 km, the carrier frequency is fixed to 2 GHz, the BS height is 25 meters and the MT height can vary from 1.5 to 22.5 m.

`3GPP_3D_UMi` - 3GPP 3D Urban Micro-Cell (3GPP TR 36.873, [9])	

`3GPP_3D_UMi_LOS`
`3GPP_3D_UMi_LOS_O2I`
`3GPP_3D_UMi_NLOS`
`3GPP_3D_UMi_NLOS_O2I`
	For typical terrestrial small-cell BSs deployed below rooftop in densely populated urban areas. Parameters cover LOS, NLOS and O2I. The maximum cell radius is about 200 m, the carrier frequency is fixed to 2 GHz, the BS height is 10 meters and the MT height can vary from 1.5 to 22.5 m.

3GPP NR model (3GPP TR 38.901) [14]	
`3GPP_38.901_Indoor_Mixed_Office`, `3GPP_38.901_Indoor_Open_Office` - 3GPP NR Indoor Office	

`3GPP_38.901_Indoor_LOS`
`3GPP_38.901_Indoor_NLOS`
	For typical indoor deployments such as WiFi or femto-cells covering carrier frequencies from 500 MHz to 100 GHz. The BS antenna height is fixed to 3 meters and the MT antenna height to 1 meter. Two office types are covered: *Mixed office* and *Open office*. They differ only in their LOS probability. The office type can be selected using '`qd_layout.set_scenario`'.

`3GPP_38.901_UMa` - 3GPP NR Urban Macro-Cell	

`3GPP_38.901_UMa_LOS`
`3GPP_38.901_UMa_LOS_O2I`
`3GPP_38.901_UMa_NLOS`
`3GPP_38.901_UMa_NLOS_O2I`
	For typical terrestrial base stations deployed above rooftop in densely populated urban areas. Parameters cover LOS, NLOS and O2I. The maximum cell radius is about 1 km, the carrier frequency can be varied from 500 MHz to 100 GHz, the BS height is 25 meters and the MT height can vary from 1.5 to 22.5 m

`3GPP_38.901_UMi` - 3GPP NR Urban Micro-Cell	

`3GPP_38.901_UMi_LOS`
`3GPP_38.901_UMi_LOS_O2I`
`3GPP_38.901_UMi_NLOS`
`3GPP_38.901_UMi_NLOS_O2I`
	For typical terrestrial base stations deployed below rooftop in densely populated urban areas. Parameters cover LOS, NLOS and O2I. The maximum cell radius is about 200 m, the carrier frequency can be varied from 500 MHz to 100 GHz, the BS height is 10 meters and the MT height can vary from 1.5 to 22.5 m

`3GPP_38.901_RMa` - 3GPP NR Rural Macro-Cell	

`3GPP_38.901_RMa_LOS`
`3GPP_38.901_RMa_LOS_O2I`
`3GPP_38.901_RMa_NLOS`
`3GPP_38.901_RMa_NLOS_O2I`
	For typical rural base stations. Parameters cover LOS, NLOS and O2I. The maximum cell radius is about 10 km, the carrier frequency can be varied from 500 MHz to 100 GHz, the BS height can vary from 10 to 150 m, and the MT height can vary from 1.5 to 22.5 m

mmMAGIC model [15]	
mmMAGIC_Indoor - mmMAGIC Indoor Office	
mmMAGIC_Indoor_LOS mmMAGIC_Indoor_NLOS	For typical indoor deployments such as WiFi or femto-cells covering carrier frequencies from 6 GHz to 100 GHz with up to 2 GHz bandwidth. The BS antenna height is fixed to 3 meters and the MT antenna height to 1 meter.
mmMAGIC_UMi - mmMAGIC Urban Micro-Cell	
mmMAGIC_UMi_LOS mmMAGIC_UMi_LOS_O2I mmMAGIC_UMi_NLOS mmMAGIC_UMi_NLOS_O2I	For typical terrestrial pico-base stations deployed below rooftop in densely populated urban areas covering carrier frequencies from 6 GHz to 100 GHz with up to 2 GHz bandwidth. The BS antenna height can vary from 6 to 10 meters and the MT antenna height is fixed to 1.5 meters.

Measurement-based parameters	
QuaDRiGa_Industrial - Industrial Indoor Scenario [16]	
QuaDRiGa_Industrial_LOS QuaDRiGa_Industrial_NLOS	For industrial-indoor deployments (e.g. for factory halls) covering carrier frequencies from 2 GHz to 6 GHz. The BS antenna height can vary from 1 to 10 meters and the MT antenna height can vary from 1 to 3 meters. The maximum diameter of the hall can be up to 150 m with a ceiling height ranging from 6 to 10 meters. This scenario supports **dual-mobility**.
QuaDRiGa_UD2D - Urban Device-to-Device Scenario	
QuaDRiGa_UD2D_LOS QuaDRiGa_UD2D_NLOS	For device-to-device communications in an outdoor urban environment. Parameters were extracted from measurements in an UMi deployment in Berlin [17]. However, the values have not be validated by measurements. The parameters are for testing only. BS and MT antenna heights are fixed to 1.5 meters. This scenario supports **dual-mobility**.
BERLIN_UMa_LOS BERLIN_UMa_NLOS	Terrestrial Urban Macrocell parameters extracted from measurements in Berlin, Germany [18].
DRESDEN_UMa_LOS DRESDEN_UMa_NLOS	Terrestrial Urban Macrocell parameters extracted from measurements in Dresden, Germany [18].

WINNER model [4, 6]	
WINNER_UMa_C2_LOS WINNER_UMa_C2_NLOS	WINNER Urban Macrocell For typical terrestrial base stations deployed above rooftop in densely populated urban areas. The max. cell radius is about 1 km.
WINNER_UMi_B1_LOS WINNER_UMi_B1_NLOS	WINNER Urban Microcell For typical terrestrial pico-base stations deployed below rooftop in densely populated urban areas. The max. cell radius is about 200 m.
WINNER_SMa_C1_LOS WINNER_SMa_C1_NLOS	WINNER Sub-Urban Macrocell For typical terrestrial base stations deployed above rooftop in sub-urban areas. The max. cell radius is about 10 km.
WINNER_Indoor_A1_LOS WINNER_Indoor_A1_NLOS	WINNER Indoor Hotspot For typical indoor deployments such as WiFi or femto-cells.
WINNER_UMa2Indoor_C4_LOS WINNER_UMa2Indoor_C4_NLOS	WINNER Urban Macrocell to Indoor For users within buildings that are connected to a terrestrial base station deployed above rooftop in densely populated urban areas.
WINNER_UMi2Indoor_B4_LOS WINNER_UMi2Indoor_B4_NLOS	WINNER Urban Microcell to Indoor For users within buildings that are connected to terrestrial pico-base stations deployed below rooftop in densely populated urban areas.

MIMOSA parameters (MIMO over Satellite) [19, 20]	
MIMOSA_10-45_LOS MIMOSA_10-45_NLOS	MIMOSA Satellite to Mobile Parameters for Urban Propagation Elevation range from 10 to 45°. Parameters were extracted from terrestrial measurement using a high-altitude platform.
MIMOSA_16-25_LOS MIMOSA_16-25_NLOS	MIMOSA Satellite to Mobile Parameters for Urban Propagation Elevation range from 16 to 25°. Parameters were extracted from terrestrial measurement using a high-altitude platform.
MIMOSA_25-35_LOS MIMOSA_25-35_NLOS	MIMOSA Satellite to Mobile Parameters for Urban Propagation Elevation range from 25 to 35°. Parameters were extracted from terrestrial measurement using a high-altitude platform.
MIMOSA_35-45_LOS MIMOSA_35-45_NLOS	MIMOSA Satellite to Mobile Parameters for Urban Propagation Elevation range from 35 to 45°. Parameters were extracted from terrestrial measurement using a high-altitude platform.

2.5 Description of the Parameter Table

The QuaDRiGa channel model is a generic model. That means, that it uses the same method for generating channel coefficients in different environments. E.g. the principal approach is exactly the same in a cellular network and in a satellite network. The only difference is the parametrization for both cases. Each environment is described by 55 individual parameters. These parameters are stored in configuration files that can be found in the subfolder named "config" in the main channel model folder. The parameters and values can be described as follows:

- **No. Clusters**
  This value describes the number of clusters. Each cluster is the source of a reflected or scattered wave arriving at the receiver. Typically there are less clusters in a LOS scenario then in a NLOS scenario. Note that the number of clusters directly influences the time needed to calculate the coefficients.

- **Path Loss (PL)**
  A common path loss (PL) model for cellular systems is the log-distance model

  $$\mathrm{PL}_{[\mathrm{dB}]} = A \cdot \log_{10} \frac{d}{d_0} + B + C \log_{10} \frac{f}{f_0} \tag{1}$$

  where A, B and C are scenario-specific coefficients. They are typically determined by measurements. d is the distance between the transmitter and the receiver, f is the carrier frequency. d_0 is the reverence distance (usually 1 m), f_0 is the reference frequency (usually 1 GHz). The path-loss exponent A typically varies between 20 and 40, depending on the propagation conditions, the base station height and other influences. In other environments such as in satellite systems, the PL does not depend on the distance but has a constant value. In this case, A would be 0.

- **Shadow Fading (SF)**
  Shadow fading occurs when an obstacle gets positioned between the wireless device and the signal transmitter. This interference causes significant reduction in signal strength because the wave is shadowed or blocked by the obstacle. It is modeled as log-normal distributed with two parameters: The standard deviation SF_σ defines the width of the distribution, i.e. the power value (in dB) above or below the distance dependent PL and the decorrelation distance SF_λ. This parameter defines how fast the SF varies when the terminal moves through the environment. E.g. a value of 87 means that when the terminal moves 87 m in any given direction, then the correlation of the value at this distance with the value at the initial position is $e^{-1} = 0.37$.

- **Delay Spread (DS)**
  The root-mean-square (RMS) delay spread is probably the most important single measure for the delay time extent of a multipath radio channel. The RMS delay spread is the square root of the second central moment of the power delay profile and is defined to be

  $$\mathrm{DS} = \sqrt{\frac{1}{P_i} \cdot \sum_{l=1}^{L} P_l \cdot (\tau_l)^2 - \left(\frac{1}{P_i} \cdot \sum_{l=1}^{L} P_l \cdot \tau_l \right)^2} \tag{2}$$

  with P_i is the total received power, P_l the cluster-power and τ_l the cluster delay.

  In order to generate the coefficients, QuaDRiGa has to generate delays for each of the multipath clusters. I.e. the total lengths of scattered paths have to be defined. This generation of delays is governed by value of the DS in a specific environment. The DS is assumed to be log-normal distributed and defined by two parameters: Its median value DS_μ and its STD DS_σ. Thus, a value for DS_μ of -6.69 corresponds to a median DS of 204 ns (Note: mean and median differ for log-normal distributions). DS_σ then defines the range of the values. E.g. $\mathrm{DS}_\sigma = 0.3$ leads to typical values in the range of $10^{-6.69-0.3} = 102$ ns to $10^{-6.69+0.3} = 407$ ns. As for the shadow fading, the decorrelation distance DS_λ defines how fast the DS varies when the terminal moves through the environment.

The delay spread is calculated from both, the delays τ_l and the path powers P_l. I.e. lager delay spreads can either be achieved by increasing the values of τ_l and keeping P_l fixed or adjusting P_l and keeping τ_l fixed. In order to avoid this ambiguity, an additional proportionality factor (delay factor) r_τ is introduced to scale the width of the distribution of τ_l. r_τ is calculated from measurement data. See Sec. 3.3 for more details.

- **Ricean K-Factor (KF)**
  Rician fading occurs when one of the paths, typically a line of sight signal, is much stronger than the others. The KF K is the ratio between the power in the direct path and the power in the other, scattered, paths. As for the DS, the KF is assumed to be log-normal distributed. The distribution is defined by its median value KF_μ and its STD KF_σ. The decorrelation distance KF_λ defines how fast the KF varies when the terminal moves through the environment.

- **Angular Spread**
  The angular spread defines the distribution of the departure- and arrival angles of each multipath component in 3D space seen by the transmitter and receiver, respectively. Each path gets assigned an azimuth angle in the horizontal plane and an elevation angle in the vertical plane. Thus we have four values for the angular spread:

  1. Azimuth spread of Departure (AsD)
  2. Azimuth spread of Arrival (AsA)
  3. Elevation spread of Departure (EsD)
  4. Elevation spread of Arrival (EsA)

  Each one of them is assumed to be log-normal distributed. Hence, we need the parameters μ, σ and λ to define the distributions. These spreads are translated into specific angles for each multipath cluster. Additionally, we assume that clusters are the source of several multipath components that are not resolvable in the delay domain. Thus, these sub-paths do not have specific delays, but they have different departure- and arrival angles. Thus, we need an additional parameter c_ϕ for each of the four angles that scales the dimensions of the clusters in 3D-space. See Sec. 3.3 for details.

- **Cross-polarization Ratio (XPR)**
  The XPR defines how the polarization changes for a multipath component. I.e. the initial polarization of a path is defined by the transmit antenna. However, for the NLOS components, the transmitted signal undergoes some diffraction, reflection or scattering before reaching the receiver.

  The XPR (in dB) is assumed to be normal distributed where μ and σ define the distribution. We translate the XPR in a polarization rotation angle which turns the polarization direction. A XPR value where a value of +Inf means that the axis remains the same for all NLOS paths. I.e. vertically polarized waves remain vertically polarized after scattering. On the other hand, a value of -Inf dB means that the polarization is turned by 90°. In case of 0 dB, the axis is turned by 45°, i.e. the power of a vertically polarized wave is split equally into a H- and V component.

The following table gives an overview of the parameters in the config files. They get converted into a structure '`qd_parameter_set.scenpar`'.

Parameter	Unit or type	Description
plpar.model (PL_model)	Text string	Selects the model for average path loss. For satellite applications the pathloss is defined by the satellite distance and is assumed to be constant for the reception are. For terrestrial cases pathloss models like "Hata" or others (e.g. WINNER pathloss models) can be selected.
plpar.A (PL_A)	dB	For satellite applications this parameter defines the average path loss and is equivalent to the "mu" of the lognormal distribution of the shadow fading.
Parameters in structure '`qd_parameter_set.scenpar`'		
Large-Scale Parameters		Those parameters describe how the large-scale parameters vary within a propagation environment.

SF_sigma SF_delta SF_lambda	dB dB/GHz meter	Those parameter describe the slow fading implemented as Lognormal distribution and filtered according to the decorrelation distance “lambda”. `SF_delta` describes the frequency dependence relative to 1 GHz.
KF_mu KF_gamma KF_sigma KF_delta KF_lambda	dB dB/GHz dB dB/GHz meter	Statistical properties of the K-factor. `KF_gamma` describes the frequency dependence of the average KF relative to 1 GHz, `KF_delta` describes the frequency dependence of the KF spread relative to 1 GHz
XPR_mu XPR_gamma XPR_sigma XPR_delta XPR_lambda	dB dB/GHz dB dB/GHz meter	The XPR is defined by the XPR-Antenna (see antenna pattern) and the XPR “environment”. The parameters describe the statistical properties of the XPR “environment”. Note: For the LOS component no XPR environment is assumed, only the XPR antenna is applied. Hence the overall XPR depends also highly on the K-factor.
DS_mu DS_gamma DS_sigma DS_delta DS_lambda	log10([s]) log10([s])/GHz log10([s]) log10([s])/GHz meter	Statistical properties of the delay spread.
AS_A_mu AS_A_gamma AS_A_sigma AS_A_delta AS_A_lambda	log10([deg]) log10([deg])/GHz log10([deg]) log10([deg])/GHz meter	Statistical properties of the azimuth of arrival spread at the receiver.
ES_A_mu ES_A_gamma ES_A_sigma ES_A_delta ES_A_lambda	log10([deg]) log10([deg])/GHz log10([deg]) log10([deg])/GHz meter	Statistical properties of the elevation of arrival spread at the receiver.
AS_D_mu AS_D_gamma AS_D_sigma AS_D_delta AS_D_lambda	log10([deg]) log10([deg])/GHz log10([deg]) log10([deg])/GHz meter	Statistical properties of the azimuth of departure spread at the transmitter.
ES_D_mu ES_D_gamma ES_D_mu_A ES_D_mu_min ES_D_sigma ES_D_delta ES_D_lambda	log10([deg]) log10([deg])/GHz log10([deg])/m log10([deg]) log10([deg]) log10([deg])/GHz meter	Statistical properties of the elevation of departure spread at the transmitter. The additional parameter `ES_D_mu_A` described the distance-dependency of the ESD relative to one meter. `ES_D_mu_min` describes the minimum allowed ESD.
Cross correlations		There are interdependencies between parameters. For example, if the K-Factor is high, the delay spread gets shorter since more power is coming from the direct component. This is expressed by the cross-correlations parameters. They can vary between -1 and 1. Negative values denote inverse correlation, e.g. a high K-Factor implies a low delay spread. Positive Value implies a positive correlation such as a high K-Factor also implies a high shadow fading. Cross-Correlations are used during the map-generation.
ds_kf asA_ds esA_ds ds_sf asA_kf esA_kf sf_kf esA_asA asA_sf esA_sf		Correlation of delay spread and K-Factor. Correlation of delay spread and azimuth of arrival angle spread. Correlation of delay spread and elevation of arrival angle spread. Correlation of delay spread and shadow fading. Correlation of K-Factor and azimuth of arrival angle spread. Correlation of K-Factor and elevation of arrival angle spread. Correlation of K-Factor and shadow fading. Correlation elevation of arrival angle spread and azimuth of arrival angle spread. Correlation of shadow fading and azimuth of arrival angle spread. Correlation of shadow fading and elevation of arrival angle spread.
Cluster Parameter		Those parameters influence the generation of the scattering-clusters and the distribution of the sub-paths within each cluster.
NumClusters	Integer	The number of clusters including LOS and (optional) ground reflection cluster. For multipath rich environments typically more clusters are used. If the LOS component is dominant a lower number of clusters is sufficient.
NumSubPaths	Integer	The number of sub-paths per cluster. The default value is 20.

SubpathMethod	Text string	Selector for the subpath generation method. The default 3GPP and WINNER model use 'legacy' model. Alternatively, the 'mmMAGIC' [15] model can be used.
PerCluster_DS PerClusterDS_gamma PerClusterDS_min	ns ns/GHz ns	The per-cluster delay spread in nanoseconds. **PerClusterDS_gamma** describes the frequency dependence of the cluster DS relative to 1 GHz. **PerClusterDS_min** describes the minimum value.
PerCluster_AS_A PerCluster_ES_A PerCluster_AS_D PerCluster_ES_D	deg deg deg deg	The azimuth of arrival angular spread of the sub-paths within one cluster. The elevation of arrival angular spread of the sub-paths within one cluster. The azimuth of departure angular spread of the sub-paths within one cluster. The elevation of departure angular spread of the sub-paths within one cluster.
LNS_ksi		Normally, cluster powers are taken from an exponential power-delay-profile. This parameter enables an additional variation of the individual cluster powers around the PDP.
r_DS		This parameter allows the mapping of delay-spreads to delays and powers for the clusters. See section 3.3.
Spatial consistency parameters		Those parameters influence the locations-dependence of the large-and small-scale parameters.
SC_lambda	meter	Decorrelation distance of all random variables used for generating the small-scale-fading.
Ground reflection parameters		Those parameters control the ground reflection.
GR_enabled GR_epsilon	0/1 complex	Enables (1) or disables (0, default) the explicit ground reflection. Fixed reflection coefficient (complex-valued) of the ground. If a value of 0 is set, the reflection coefficient is determined by a frequency-dependent model. See: [13]

2.6 Data Exchange Formats

2.6.1 QuaDRiGa Array Antenna Exchange Format (QDANT)

In QuaDRiGa, antenna data is described over a surface as a function of position relative to the antenna. Far-field data is mapped to spherical surfaces from which directivity, polarization and patterns are calculated. A reference for antenna measurements and coordinate systems can be found in [21]. QuaDRiGa uses the Polar-Spherical (Theta-Phi) coordinate system, where the antenna coordinate system has two angles and two poles. The elevation angle θ is measured relative to the pole axis. A complete circle will go through each of the two poles, similar to the longitude coordinate in the world geodetic system (WGS). The azimuth angle ϕ moves around the pole, similar to the latitude in WGS. Thus, the antenna is defined in *geographic* coordinates.

Polarization is described in a polar-spherical polarization basis in which the the pole of the polarization basis is placed along the z-axis. The electric field is resolved onto three vectors which are aligned to each of the three spherical unit vectors $\hat{\mathbf{e}}_\theta$, $\hat{\mathbf{e}}_\phi$ and $\hat{\mathbf{e}}_r$ of the coordinate system. In this representation, $\hat{\mathbf{e}}_r$ is aligned with the propagation direction of a path. In the far-field of an antenna, there is no field in this direction. Thus, the radiation pattern consists of two components, one is aligned with $\hat{\mathbf{e}}_\theta$ and another is aligned with $\hat{\mathbf{e}}_\phi$.

The QuaDRiGa array antenna exchange format is a file format used to store antenna pattern data in XML and load them into QuaDRiga. This section contains a reference for all XML elements defined in QDANT v2.2.0. Because QDANT is an XML grammar and file format, tag names are case-sensitive and must appear exactly as shown here. An example of a 2-element array antenna with perfect cross-polarization is given as:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<qdant xmlns="http://www.quadriga-channel-model.de">
<layout>1</layout>
<arrayant id="1">
    <name>Simple XPOL</name>
    <CenterFrequency>2600000000</CenterFrequency>
    <NoElements>2</NoElements>
    <ElementPosition>0,0,0 0,0,0</ElementPosition>
    <ElevationGrid>-90 0 90</ElevationGrid>
    <AzimuthGrid>-180 -90 0 90 180</AzimuthGrid>
    <CouplingAbs>1,0 0,1</CouplingAbs>
    <CouplingPhase>0,0 0,0</CouplingPhase>
    <EthetaMag el="1">
    0 0 0 0 0
    0 0 0 0 0
    0 0 0 0 0
    </EthetaMag>
    <EthetaPhase el="1">
    0 0 0 0 0
    0 0 0 0 0
    0 0 0 0 0
    </EthetaPhase>
    <EphiMag el="2">
    0 0 0 0 0
    0 0 0 0 0
    0 0 0 0 0
    </EphiMag>
    <EphiPhase el="2">
    0 0 0 0 0
    0 0 0 0 0
    0 0 0 0 0
    </EphiPhase>
</arrayant>
</qdant>
```

qdant		
Description	The root element of a QDANT file. This element is required. It follows the XML declaration at the beginning of the file. A basic <`qdant`> element contains 0 or 1 <`layout`> element and 1 or more <`arrayant`> elements.	
Attributes	xmlns	Namespace declaration *(required)*. It is possible to use a prefix to avoid name conflicts when embedding QuaDRiGa antennas in other XML formats. When using a prefix in XML, a namespace for the prefix must be defined. The namespace can be defined by an `xmlns` attribute in the start tag of an element (see example below).
Elements	layout	A layout describes the organization of multiple array antennas into an object array *(optional)*. The containing integer numbers must match the `id` attributes of the <`arrayant`> elements. If objects are arranged as a matrix, <`layout`> contains column vectors where the elements are separated by a comma and the multiple vectors are separated by space.
	arrayant	The definition of the array antenna *(required)*.

Example using a prefix:

```
<qdant xmlns:qdant="http://www.quadriga-channel-model.de">
<qdant:layout>1,2 1,2 2,1</qdant:layout>
<qdant:arrayant id="1">
    <!-- Array antenna definition for object 1 in the layout -->
</qdant:arrayant>
<qdant:arrayant id="2">
    <!-- Array antenna definition for object 2 in the layout -->
</qdant:arrayant>
</qdant>
```

arrayant		
Description	This element describes a single array antenna.	
Attributes	id	If multiple array antennas are stored in the same file, each antenna must be identified by an unique ID. Array antenna IDs are integer numbers. Each array antenna ID must be present in the <`layout`>.
Elements	name	User defined text identifying the array antenna (optional).
	CenterFrequency	Center frequency in [Hz] for which the antenna patterns are defined (optional). If this parameter is not given, a default value of 300 MHz is assumed. This leads to the element positions being expressed as multiples of the wavelength.
	NoElements	Number of elements in an array antenna (required for multi-element antennas).
	ElementPosition	Position [x,y,z] of the elements in units of [meters] relative to the phase center of the antenna (optional). The x-axis goes from west to east; the y-axis goes from south to north; the z-axis goes from down to up. Position vectors of multiple elements are separated by space. To set the `ElementPosition` for multiple elements, you must define `NoElements` first.
	ElevationGrid	Sampling grid in [degrees] of the antenna patterns in elevation direction. Values range from -90 (pointing down) to 90 (pointing up). The 0 degree value points to the horizon. This XML-element is always required.
	AzimuthGrid	Sampling grid in [degrees] of the antenna patterns in azimuth direction. Values range from -180 (pointing west) to 180 (also pointing west). The 0 degree value points east. Counting is anti-clockwise. The 90 degree value points north, the -90 degree value points south. This XML-element is always required.
	CouplingAbs	Absolute values of the coupling matrix (optional). Column vectors are separated by a space sign, entries within a column vector are separated by commas. `CouplingAbs` cannot be defined without defining `NoElements` first.
	CouplingPhase	Phases of the coupling matrix in [degree] (optional). Column vectors are separated by a space sign, entries within a columns vector are separated by commas. `CouplingPhase` cannot be defined without defining `CouplingAbs` first.

	EthetaMag	Magnitude in [dB] of the electric field aligned with the $\hat{\mathbf{e}}-\theta$ vector of the spherical coordinate system. One line contains all azimuth values for one elevation angle. The order is given by `AzimuthGrid` and `ElevationGrid`. If `NoElements` is greater than 1, an additional attribute `<EthetaMag el="[no]">` must be provided. In the first example, the first line contains the values for the -90 degree elevation angle (down) and the last line contains the values for the 90 degree elevation (up). If there is no field in θ-direction, this XML-element can be omitted.
	EthetaPhase	Phase in [degree] of the electric field aligned with the $\hat{\mathbf{e}}-\theta$ vector. One line contains all azimuth values for one elevation angle. The order is given by `AzimuthGrid` and `ElevationGrid`. If `NoElements` is greater than 1, an additional attribute `<EthetaPhase el="[no]">` must be provided. `EthetaPhase` cannot be defined without defining `EthetaMag` first. If all phases are 0, this XML-element can be omitted.
	EphiMag	Magnitude in [dB] of the electric field aligned with the $\hat{\mathbf{e}}-\phi$ vector of the spherical coordinate system. One line contains all azimuth values for one elevation angle. The order is given by `AzimuthGrid` and `ElevationGrid`. If `NoElements` is greater than 1, an additional attribute `<EphiMag el="[no]">` must be provided. If there is no field in ϕ-direction, this XML-element can be omitted.
	EphiPhase	Phase in [degree] of the electric field aligned with the $\hat{\mathbf{e}}-\phi$ vector. One line contains all azimuth values for one elevation angle. The order is given by `AzimuthGrid` and `ElevationGrid`. If `NoElements` is greater than 1, an additional attribute `<EphiPhase el="[no]">` must be provided. `EphiPhase` cannot be defined without defining `EphiMag` first. If all phases are 0, this XML-element can be omitted.

2.6.2 QuaDRiGa Layout Exchange Format (KML)

QuaDRiGa layouts can be loaded from KML files. KML (short for Keyhole Markup Language) is a file format used to store geographic data and visualize it in an Earth browser such as Google Earth. KML is an international standard maintained by the Open Geospatial Consortium, Inc. (OGC). In order to use KML to exchange QuaDRIGa layouts, the file must follow some specific formatting conventions which are detailed in this section. Each KML layout can be displayed in Google Earth and it is possible to draw QuaDRiGa layouts in Google Earth without knowledge of MATLAB or the QuaDRiGa API.

The following example provides a parameterization of QuaDRiGa using KML. There is one transmitter (`tx_HHI`), one mobile receiver (`rx_Car`) and a scenario definition for the Rx (`seg_LOSonly`). All of these are defined by `Placemark` elements which will also be shown by Google Earth. Additional model parameters, such as the center frequency, or the which antennas to use, are defined by `ExtendedData` elements. Those will be shown by Google Earth, but not processed any further.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
<name>test.kml</name>
<Folder>
    <name>Test Layout</name>
    <ExtendedData>
        <Data name="CenterFrequency"><value>2600000000</value></Data>
        <Data name="UpdateRate"><value>0.001</value></Data>
    </ExtendedData>
    <Placemark>
        <name>tx_HHI</name>
        <ExtendedData>
            <Data name="Antenna"><value>antenna.qdant</value></Data>
        </ExtendedData>
        <Point>
            <extrude>1</extrude><altitudeMode>relativeToGround</altitudeMode>
            <coordinates>13.3249472,52.5163194,65</coordinates>
        </Point>
    </Placemark>
    <Placemark>
        <name>rx_Car</name>
        <ExtendedData>
            <Data name="Antenna"><value>dipole.qdant</value></Data>
            <Data name="Time"><value>10</value></Data>
        </ExtendedData>
        <LineString>
            <altitudeMode>relativeToGround</altitudeMode>
            <coordinates>13.325849458441,52.5163194,1.5
                13.326058636494,52.516192120779,1.5</coordinates>
        </LineString>
    </Placemark>
    <Placemark>
        <name>seg_LOSonly</name>
        <Point>
            <altitudeMode>relativeToGround</altitudeMode>
            <coordinates>13.325849458441,52.5163194,1.5</coordinates>
        </Point>
    </Placemark>
</Folder>
</Document>
</kml>
```

kml		
Description	The root element of a KML file. This element is required. It follows the XML declaration at the beginning of the file. The `<kml>` element may also include the namespace for any external XML schemas that are referenced within the file, e.g. the antenna definitions.	
Attributes	xmlns	Namespace declaration *(required)*. It is possible to store the antenna patterns within the KML file. In this case, a namespace for the antennas must be defined. The namespace can be defined by an additional **xmlns** attribute in (see example below).
Elements	Document	Container for additional elements *(required)*.

Document		
Description	A Document is a container. It may contain the layout definition, embedded antenna patterns and shared styles. Styles are not processed by QuaDRiGa, but are used to visualize data in Google Earth.	
Elements	name ExtendedData Folder	The name of the Document, usually the fila name. *(optional)* Embedded antenna patterns. *(optional)* Container or the QuaDRiGa Layout. *(required)*

Example using embedded antenna patterns:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:qdant="http://www.quadriga-channel-model.de">
<Document>
    <name>test.kml</name>
    <ExtendedData>
        <qdant:layout>1,2</qdant:layout>
        <qdant:arrayant id="1">
            <!-- Array antenna definition for object 1 in the layout -->
        </qdant:arrayant>
        <qdant:arrayant id="2">
            <!-- Array antenna definition for object 2 in the layout -->
        </qdant:arrayant>
    </ExtendedData>
    <Folder>
        <!-- Layout definition -->
    </Folder>
</Document>
</kml>
```

Folder		
Description	A Folder is used to arrange other features hierarchically (Folders, Placemarks, etc.). For QuaDRiGa, the KML file may contain a single folder in which the QuaDRiGa layout is defined.	
Elements	name	The name of the Folder *(optional)*. This name will be used as the layout name when loading the KML file in QuaDRiGa. If no name is specified, "Layout" will be used as default name name.
	ExtendedData	List of simulation settings. *(optional, recommended)*
	Placemark	Location information for transmitters, receivers or segments. *(required)*

Placemark		
Description	A Placemark is a Feature with associated Geometry. It can either define a point or a track (LineString), i.e. a list of consecutive points. In Google Earth, a Placemark appears as a list item in the Places panel. A Placemark with a Point has an icon associated with it that marks a point on the Earth in the 3D viewer. In QuaDRiGa, `Placemark` elements define the positions of transmitters, receivers and the start-points of segments along a track.	
Usage	Transmitter	If a Placemark defines a transmitter in QuaDRiGa, the Placemark name must start with "`tx_`" followed by the transmitter name (the underscore "_" character is not allowed in the name). When loaded by QuaDRiGa, the transmitter name will be assigned to the list of transmitters without the "`tx_`" in the name. All transmitters in a layout must have unique names. Using the same transmitter name more than once will lead to an error. Transmitters can be either static (using a Point element) or mobile (using a LineString element). There must be at least one transmitter in the layout. Transmitters are processed in the order they are defined in the KML file.
	Receiver	If a Placemark defines a receiver in QuaDRiGa, the Placemark name must start with "`rx_`" followed by the receiver name. All receivers in a layout must have unique names. Receivers can be either static (using a Point element) or mobile (using a LineString element). There must be at least one receiver in the layout. Receivers are processed in the order they are defined in the KML file.
	Segment	Segments assign propagation conditions to a radio link. Segment Placemarks are only allowed to contain a Point element. They must be placed close to a receiver Placemark and there must be at least one segment for each receiver in the Layout. If the receiver contains a track (LineString), multiple segments can be defined for a receiver. In this case, the closest position (projection) on the track is used as segment start point. If a Placemark defines a segment, the Placemark name must start with "`seg_`" followed by a list of scenario names, one for each transmitter. The scenario names equal the filename of configuration files in the `config` folder of the QuaDRiGa installation (without the ".conf" ending), e.g. "`seg_3GPP_3D_UMa_LOS`" would assign the urban-macrocell scenario to a segment. Scenarios for multiple transmitters are separated by a ":", e.g. "`seg_3GPP_3D_UMa_LOS:Freespace`". The order of segment Placemarks in the KML file is irrelevant, only the proximity to actual receiver positions counts. However, the order of the transmitters in the KML file defines in which order the scenario names must be given.
Elements	name	The name of the Placemark *(required)*. In QuaDRiGa, the Placemark name defines if the Placemark is for a transmitter, a receiver or a segment. It also defines the subsequent name of the transmitters and receivers and it associates segments with propagation conditions.
	ExtendedData	Additional attributes such as orientation and antennas. *(optional)*
	Point	A geographic location defined by longitude, latitude, and altitude.
	LineString	A connected set of line segments defining a track.

Point		
Description	A geographic location defined by longitude, latitude, and altitude. When a Point is contained by a Placemark, the point itself determines the position of the Placemark's name and icon in Google Earth. In QuaDRiGa, Points are used to define static transmitters, static receivers or segment start points.	
Elements	extrude	Boolean value. Specifies whether to connect the point to the ground with a line in Google Earth. This element is ignored by QuaDRiGa. *(optional)*
	altitudeMode	Specifies how altitude components in the <`coordinates`> element are interpreted *(optional)*. **clampToGround** ignores the altitude specification in Google Earth, but sets the altitude of the element relative to the actual ground elevation in QuaDRiGa. **relativeToGround** (default in QuaDRiGa) sets the altitude of the element relative to the actual ground elevation of a particular location. **absolute** sets the altitude of the coordinate relative to sea level, regardless of the actual elevation of the terrain beneath the element.
	coordinates	A single tuple consisting of floating point values for longitude, latitude, and altitude (in that order). *(required)*. Longitude and latitude values are in degrees, where longitude $\geq$ -180 and $\leq$ 180, latitude $\geq$ -90 and $\leq$ 90, and altitude values are in meters. Do not include spaces between the three values that describe a coordinate.

LineString		
Description	Defines a connected set of line segments. In QuaDRiGa, LineStrings are used to define mobile transmitters or receivers. A LineString must contain a minimum number of two coordinates.	
Elements	extrude	When a LineString is extruded, the line is extended to the ground, forming a polygon that looks somewhat like a wall or fence in Google Earth. This element is ignore by QuaDRiGa. *(optional)*
	altitudeMode	Specifies how altitude components in the <`coordinates`> element are interpreted *(optional)*. **clampToGround** ignores the altitude specification in Google Earth, but sets the altitude of the element relative to the actual ground elevation in QuaDRiGa. **relativeToGround** (default in QuaDRiGa) sets the altitude of the element relative to the actual ground elevation of a particular location. **absolute** sets the altitude of the coordinate relative to sea level, regardless of the actual elevation of the terrain beneath the element.
	coordinates	Two or more coordinate tuples, each consisting of floating point values for longitude, latitude, and altitude *(required)*. Longitude and latitude values are in degrees, where longitude $\geq$ -180 and $\leq$ 180, latitude $\geq$ -90 and $\leq$ 90, and altitude values are in meters. Insert a space between tuples. Do not include spaces within a tuple.

ExtendedData		
Description	The ExtendedData element offers techniques for adding custom data to a KML Feature (Placemark, Document, Folder, etc.). This is used to assign configuration parameters to QuaDRiGa that are not processed by Google Earth. This can be used in three ways: • Embed antenna patterns (by adding an ExtendedData element to the **Document** element) • Assign QuaDRiGa simulation parameters (by adding an ExtendedData element to the **Folder** element) • Provide additional setting for transmitters, receivers and segments (by adding an ExtendedData element to the **Placemark** element) Antenna patterns are embedded by referring to QDANT elements defined in another namespaces and by referencing the external namespace within the KML file (see example above). Simulation parameters and additional setting for transmitters and receivers are given as untyped data/value pairs using the Data element, e.g. "`<Data name="CenterFrequency"><value>2600000000</value></Data>`". A list of allowed settings and their meaning is given below.	
Sim. settings	CenterFrequency	Center frequency in units of [Hz] (floating point value). For multi-frequency simulations, multiple frequency values may be given separated by commas.

	UpdateRate	The update rate of the channel coefficients in units of seconds (floating point value). For example, a value of 0.01 means that channel coefficients are generated every 10 milliseconds. If the parameter is given, each track (i.e. `Placemark` with `LineString` element) must contain the `Time` parameter which specifies how long the it takes for the Tx/Rx to traverse the track. If `UpdateRate` is not given, channel coefficients are generated at the exact coordinates specified in the `LineString` or `Point` elements of the KML file.
	SampleDensity	The sampling density describes the number of samples per half-wave length (floating point value) for the highest center frequency. This parameter is only useful if `UpdateRate` is given as well. Channels are generated at the given `SampleDensity` and then interpolated to match the requested `UpdateRate`. The minimum sample density must be 1 for static transmitters and 2 for mobile transmitters. Smaller values may be given, but the Doppler characteristics of the channel will be incorrect. The default value is 2.5.
	AbsoluteDelays	Boolean value. By default (0), path delays are calculated such that the LOS delay is normalized to 0 seconds. By setting `AbsoluteDelays` to 1, the absolute path delays are included at the output of the model.
	RandInitPhase	Boolean value. By default (1), each path is initialized with a random phase (except the LOS path and the optional ground reflection). Setting `RandInitPhase` to 0 disables this function. In this case, each path gets initialized with a zero-phase.
	Baseline3GPP	Boolean value. This enables (1) or disables (0) the 3GPP baseline model. By default (0), the 3GPP baseline model is disabled and enhanced QuaDRiGa features are used. This option uses spherical waves at both ends, the transmitter and the receiver. This method uses a multi-bounce model where the departure and arrival angles are matched such that the angular spreads stay consistent. Setting `Baseline3GPP` to 1 enables the 3GPP baseline model and shortens the computation time significantly (good for testing purposes and 3GPP compliant simulations). However, the large-scale parameters (departure and arrival angles, shadow fading, delays, etc.) are not updated in this case, phases at the array antennas are calculated by a planar wave approximation, spatial consistency is not available, multi-frequency simulations are not supported, and no polarization rotation is calculated. This may lead to incorrect results for longer tracks.
	ProgressReport	Boolean value. By default (1), a progress report is generated and printed to the command line. Setting `ProgressReport` to 0 will disable this function.
	AutoCorrFcn	String. The autocorrelation function for generating correlated model parameters. An autocorrelation function (ACF) is a description of the correlation vs. distance. This function is approximated by a Fourier series. The coefficients of the series are used to generate spatially correlated random variables. There are 3 ACF types that can be selected. The coefficients are precomputed for 150, 300, 500, and 1000 sinusoids. The default is `Comb300`. • Exponential ACF (`Exp150`, `Exp300`, `Exp500`, `Exp1000`) • Gaussian ACF (`Gauss150`, `Gauss300`, `Gauss500`, `Gauss1000`) • Combined Gaussian and Exponential ACF (`Comb150`, `Comb300`, `Comb500`, `Comb1000`)
	Pairing	A list of tuples describing the TX-RX links for which channels are created (unsigned integer values, starting at 1). The first value corresponds to the transmitter and the second value to the receiver in the order they are defined in the KML file. Insert a space between tuples. Do not include spaces within a tuple.
	SplitSegments	This parameter controls the splitting of long segments into sub-segments with the same scenario definition. Radio parameters within a segment are considered static. Hence, when segments are too long (i.e. longer than 20 meters in typical setups), the radio channel gets unrealistic. Segment splitting fixes this. `SplitSegments` is a tuple of 4 values (floating point numbers) defining: 1. min. length of a sub-segment (e.g. 10 m) 2. max. length of the sub-segment; must be $>$ 2·min. (e.g. 30 m) 3. average length of the sub-segment (e.g. 15 m) 4. standard-deviation of a sub-segment (e.g. 5 m) If `SplitSegments` is not defined, segment splitting is disabled.

	ReferenceCoord	A single tuple consisting of floating point values for longitude and latitude at which the origin (0,0,0) of the metric Cartesian coordinate system used by QuaDRiGa is placed. If this value not provided, the origin is placed in the middle of all coordinate tuples given in the KML file.
Tx/Rx settings	Antenna	The antenna patterns to be used for the simulation (string). If the antennas are stored in an external QDANT file, place the filename here. If there are multiple antennas stored in the file, indicate which antenna to use by a colon followed by the antenna number, e.g. "`antennas.qdant:2`". The number must match the ID in the <`arrayant`> element. For multi-frequency simulations, each frequency band might use a different antenna. Those need to be separated by a comma, e.g. "`ant2GHz.qdant:1,ant10GHz.qdant:2`". If the antennas are embedded in the KML file, simply indicate the antenna by a colon followed by the antenna ID, e.g. "`:1,:3`". If no antenna is provided, an omni-directional antenna with vertical polarization is used by default.
	Time	The time in seconds needed to traverse a track or LineString (floating point number). This value is required for each track when an `UpdateRate` is given in the simulation parameters. If both Tx and Rx are mobile, the corresponding tracks must have the same `Time` value. It is also possible to simulate variable speeds of a terminal. in this case, you can provide a list of time points starting with 0, e.g. "`0,5,10`", and the corresponding positions on the track using the `Distance` parameter.
	Distance	List of distance values of a terminal relative to the beginning of a track in [meters] (floating point numbers). This parameter is only used in conjunction with the `Time` parameter. It needs to have the same number of elements, for example "`0,1,10`". The largest value cannot exceed the length of the track.
	Bank	The "bank angle" in [degrees], i.e., the orientation around an axis drawn through the body of the vehicle from tail to nose in the normal direction of movement (floating point numbers). Positive rotation is clockwise (seen from the pilot/drivers perspective). `Bank` can be a single value or a list of values having the same number of elements as there are coordinates in a `LineString`. If `Bank` is not provided, the default value of 0 degrees is used.
	Tilt	The "tilt angle" in [degrees], i.e. the vertical (tilt) angle relative to the horizontal plane (floating point number). Positive values point upwards. `Tilt` can be a single value or a list of values having the same number of elements as there are coordinates in a `LineString`. If `Tilt` is not provided, the angle is calculated from consecutive coordinates in the `LineString`. For static terminals, the default value of 0 degrees is used.
	Heading	The bearing or "heading angle", in mathematic sense (degrees, floating point number). Heading is used to describe the direction an object is pointing. In contrast, the course angle refers to the direction an object is actually moving. East corresponds to 0 degrees, and the angles increase counter-clockwise, so north is at 90 degree degrees, south is -90 degrees, and west is at 180 degrees. `Heading` can be a single value or a list of values having the same number of elements as there are coordinates in a `LineString`. If `Heading` is not provided, the angle is calculated from consecutive coordinates in the `LineString`. For static terminals, the default value of 0 degrees (pointing east) is used.
Segment settings	Track	Name of the associated track (*optional*), e.g., "rx_Rx0001".
	Index	The snapshot index of the Rx-track to which the segment should be assigned (*optional*). Providing the snapshot index and track name disables the proximity projection and assigns the segment directly to the Rx track.
Elements	Data	Untyped name/value pair. The name attribute is used to identify the data pair within the KML file. Each <`Data`> element must contain a <`value`> element that contains the actual data.

description	
Description	User-supplied content in a KML file. Google Earth does not allow to edit <`ExtendedData`> elements. An alternative method to configure QuaDRiGa simulation parameters and additional setting for transmitters and receivers is to write them to the <`description`> field. This can be used in two ways: • Assign QuaDRiGa simulation parameters (by adding an description element to the **Folder** element) • Provide additional setting for transmitters and receivers (by adding an description element to the **Placemark** element) Antenna patterns cannot be embedded this way and need to be provided in an external file. Simulation parameters and additional setting for transmitters and receivers are given as data/value pairs separated by an equal "=" sign, e.g. "`CenterFrequency = 2600000000`". Multiple parameters must be separated by a new line. A list of allowed settings and their meaning is in the description of the `ExtendedData` element. If `ExtendedData` and `description` elements are present in the same `Folder` or `Placemark`, values provided by `description` have preference.

Example using the description element to set QuaDRiGa parameters:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document><name>test.kml</name>
<Folder>
    <name>Test Layout</name>
    <description>CenterFrequency = 3500000000
SampleDensity = 2.5</description>
    <ExtendedData>
        <Data name="CenterFrequency"><value>2600000000</value></Data>
        <Data name="UpdateRate"><value>0.001</value></Data>
    </ExtendedData>
    <!-- Additional Placemarks -->
</Folder>
</Document>
</kml>
```

This would set the center frequency to 3.5 GHz, the sample density to 2.5, and the channel update rate to 1 ms.

3 Technical Documentation

The QuaDRiGa channel model has two main objectives:

1. Provide an open-source reference implementation of the baseline 3GPP channel models
2. Implement modeling concepts and ideas that go beyond 3GPP to support the more realistic channel simulations

The first objective covers the baseline parts of the 3GPP-3D model [9] and the 3GPP-NR model [10], but not the optional features. These baseline models have been calibrated and the results are presented in chapter 5 of the documentation. The specific approach to achieve the second objective is laid out in this chapter. Some of the optional 3GPP features (such as multi-frequency simulations and spatial consistency) have been implemented with modifications as part of this approach. Hence, 3GPP compliance cannot be guaranteed.

Geometry-based stochastic channel models (GSCMs) such as the 3GPP-SCM [22], the WINNER model [4], the European Cooperation in Science and Technology (COST) model [23] and the 3GPP-3D channel model [24] are important tools to validate new concepts in mobile communication systems. Early models such as the 3GPP-SCM [22], its extensions [25, 26], and the WINNER model [4] are based on a two-dimensional (2-D) modeling approach. However, Shafi et al. [27] pointed out the importance of a three-dimensional (3-D) extension when studying the effects of cross-polarized antennas on the MIMO capacity. This was taken up in the WINNER+ project where the parameter tables were completed with the elevation component [6]. 3-D propagation was also incorporated into other models such as the COST model [28] or mobile-to-mobile propagation models [29]. These later models share similar ideas which are incorporated into the new model outlined in this chapter.

A second aspect of major importance for various propagation environments is polarization. Multiple polarizations can be exploited to increase the number of spatial degrees of freedom especially when using compact antennas with a limited amount of elements [30, 31]. First attempts to include polarization effects into the SCM were made by Shafi *et al.* [27] who extended the simple 2-D antenna pattern of the SCM to a dual-polarized 3-D pattern. This method was then also adopted for the WINNER model [32]. However, Shafi *et al.*'s approach did not include a geometry-based method to calculate the cross-polarization effects. Instead, the XPR was incorporated statistically where the parameters were derived from measurements. This statistical approach leads to correct results for the cross-polarization discrimination (XPD)[3] in case of a well-balanced statistical mixture between LOS and NLOS scenarios in an indoor environment. However, the distribution of singular values, which is a better metric for characterizing the multi-stream capabilities of MIMO channels, was not considered. Zhou *et al.* [34] already indicated that it might be preferable to model the channel XPR by a rotation matrix. Later on, Quitin *et al.* [35] introduced an analytical channel model that correctly takes the antenna orientation into account. However, this method is limited to azimuthal propagation only (*i.e.*, no elevation angles are supported) and it does not support arbitrary antenna characteristics. It is discussed later in this chapter that the WINNER approach, which was adopted by all succeeding models, has great similarities with the Jones calculus, a method for handling polarized electromagnetic waves in the field of optics [36]. A new method to incorporate the polarization effects based on the Jones calculus is proposed in Section 3.5.

Another prerequisite for *virtual field trials* is the continuous time evolution of channel traces. Xiao *et al.* [25] added short-term time evolution to the SCM which was afterwards incorporated into an official SCM extension [26]. The idea is to calculate the position of the last-bounce scatterers (LBSs) based on the arrival angles of individual multipath components. Then, when the MT is moving, the arrival angles, delays and phases are updated using geometrical calculations. However, the WINNER-II model did not incorporate this technique and so did not the ITU, WINNER+, and 3GPP-3D model. Hence, all those models do not support time evolution beyond the scope of a few milliseconds which restricts the mobility of the MTs

[3]Following the definition in [33], the term cross polarization ratio (XPR) is used for the polarization effects in the channel. Combining the XPR with imperfect antennas yields the cross-polarization discrimination (XPD).

to a few meters. The COST model [23] incorporates time evolution by introducing groups of randomly placed scattering clusters that fade in and out depending on the MT position. However, despite the effort that was made to parameterize the model [37, 38], it still lacks sufficient parameters in many interesting scenarios. Czink *et al.* [39] introduced a simplified method that fades the clusters in and out over time. The cluster parameters were extracted from measurements, and the model is well suited for link-level simulations. However, this *random cluster model* cannot be used for system-level scenarios because it does not include geometry-based deployments. Nevertheless, the ideas presented by [39] led to more research on the birth and death probability as well as the lifetime of individual scattering clusters [40]. Wang *et al.* [41] then proposed a model for non-stationary channels that allows the scattering clusters to be mobile.

This chapter describes an extension of the WINNER model [4] where time evolution, geometric polarization and 3-D propagation effects such as spherical waved are incorporated. A reference implementation in MATLAB/Octave is available as open source [42]. The modeling approach consists of two steps: a stochastic part generates so-called large-scale parameters (LSPs) (*e.g.*, the delay and angular spreads) and calculates random 3-D positions of scattering clusters. Both ends of the communication link can be mobile, i.e. moving base stations (BSs) and mobile terminals (MTs) are supported. Scattering clusters are fixed and the time evolution of the radio channel is deterministic. Different positions of the MT lead to different arrival angles, delays and phases for each multipath component (MPC). Longer sequences are generated by transitions between channel traces from consecutive initializations of the model. This allows the MTs to traverse different scenarios, *e.g.*, when moving from indoors to outdoors.

Figure 6 gives an overview of the modeling steps. The network layout (*i.e.*, the positions of the BSs, antenna configurations and downtilts), the positions and trajectories of the MTs, and the propagation scenarios need to be given as input variables. The channel coefficients are then calculated in seven steps which are described in detail in Sections 3.1 to 3.8 of this chapter. A quick summary of the procedure for generating the channel coefficients is given in the following list.

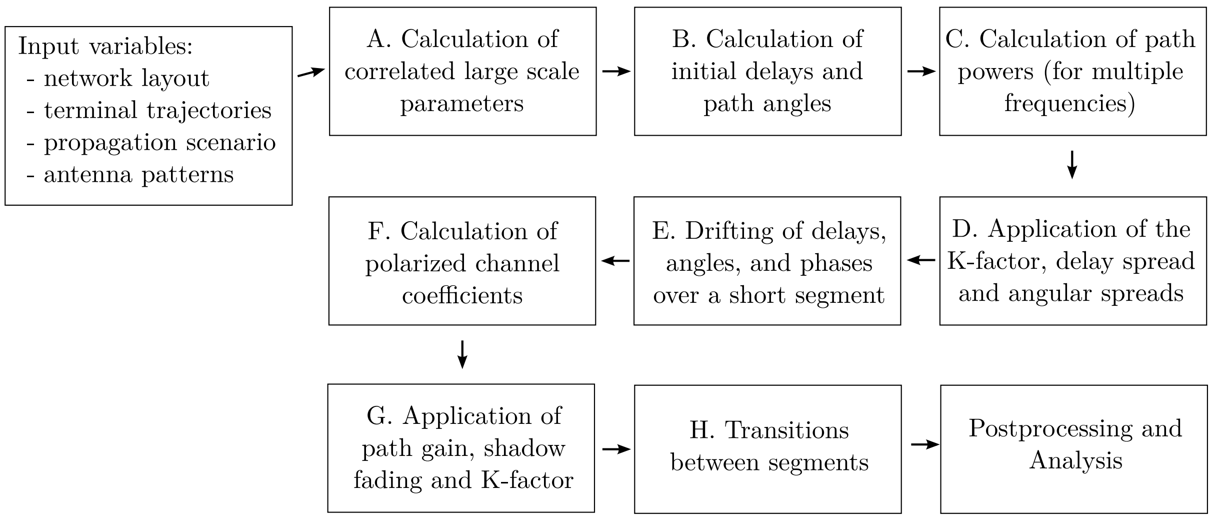

Input variables:
- network layout
- terminal trajectories
- propagation scenario
- antenna patterns
A. Calculation of correlated large scale parameters
B. Calculation of initial delays and path angles
C. Calculation of path powers (for multiple frequencies)
D. Application of the K-factor, delay spread and angular spreads
E. Drifting of delays, angles, and phases over a short segment
F. Calculation of polarized channel coefficients
G. Application of path gain, shadow fading and K-factor
H. Transitions between segments
Postprocessing and Analysis

Figure 6: Steps for the calculation of time-evolving channel coefficients

A. Calculation of correlated large scale parameters
   The first step ensures that the LSPs are consistent. As the name implies, these parameters (delay and angular spreads, K-factor and shadow fading) do not change rapidly. Closely spaced MTs will thus experience similar propagation effects.

B. Calculation of initial path delays and angles
   Once the LSPs are known, the spatially-correlated fast-fading channels are calculated for each MT separately. This step takes the specific values of the delay spread and the four angular spreads from step A and translates them into a set of multipath components, each having a specific delay and a

departure direction at the transmitter (TX) and an arrival direction at the receiver (RX). It is assumed here, that the propagation path always consists of multiple scatterers and there is no geometric relation between delays and angles.

C. Calculation of the path powers
Each MPC gets assigned specific power values for one or more carrier frequencies. The delay and angular values from step B remain unchanged during that process.

D. Application of the K-factor, delay spread and angular spreads
The initial path delays, angles and powers from steps B and C are updated to accurately reflect the KF, DS and angular spreads (ASs). As a result, it is possible to calculate these statistical measures of the channel from the channel coefficients.

E. Drifting of delays, angles, and phases over a short segment
This step incorporates mobility and spherical waves at both ends of the communication link. Given the angles and delays (*i.e.*, the output of step D), it is possible to calculate the exact position of the first-bounce scatterer (FBS) and the last-bounce scatterer (LBS), *i.e.*, the first and last reflection of a MPC. Then, when the MTs move to a different location, the scatterer positions of all MPCs are kept fixed and the delays and directions are updated. This also leads to an update of the phases of the MPCs which reflect the correct Doppler shift when the MT is moving because the length of a propagation path changes in a deterministic manner.

F. Calculation of polarized channel coefficients
This step takes care of the antenna and polarization effects. The antennas are described by their 3-D far field radiation patterns in a polar-spheric representation [43]. However, those patterns are given in an antenna-specific local coordinate system. Thus, this step includes a method to rotate the antennas to match the MT and BS orientations defined in the global coordinate system (GCS) at the input of the model. Then, additional changes in the polarization might occur during scattering of a MPC. The resulting effects are handled by a method inspired by the Jones calculus [36] where successive linear transformations are used to calculate the polarization state of a MPC.

G. Application of path gain, shadow fading and K-Factor
In this step, the remaining LSPs from step A, *i.e.*, the distance-dependent path gain and the shadow fading, are applied to the channel coefficients. When the MT position changes during drifting in step E, the Ricean K-factor at the new location might be different. This is taken into account here as well.

H. Transitions between segments
Longer sequences of channel coefficients need to consider the birth and death of scattering clusters as well as transitions between different propagation environments. This is addressed by splitting the MTs trajectories into *segments*. A segment can be seen as an interval in which the LSPs do not change considerably and where the channel maintains its wide sense stationary (WSS) properties. Channel traces are then generated independently for each segment (*i.e.*, steps B-G). Those individual traces are combined into a longer sequence in the last step.

Time evolution requires a more detailed description of the mobility of the terminals compared to previous models. This is done by assigning tracks, *i.e.*, ordered lists of positions, to each MT. In reality, this may include accelerations, decelerations, and MTs with different speeds, *e.g.*, pedestrian and vehicular users. However, to minimize the computational overhead and memory requirements, channel coefficients are calculated at a constant sample rate that fulfills the sampling theorem

$$f_T \geq 2 \cdot B_D = 4 \cdot \max|\Delta f_D| = 4 \cdot \frac{\max|v|}{\lambda_c}, \tag{3}$$

where B_D is the width of the Doppler spectrum, Δf_D is the maximum frequency change due to the velocity v, and λ_c is the carrier wavelength. Thus, the appropriate sampling rate is proportional to the maximum speed of the MT. Since it is sometimes useful to examine algorithms at different speeds, it is undesirable to fix the sampling rate in advance as the speed is then fixed as well. To overcome this problem, channel coefficients are calculated at fixed positions with a sampling rate f_S measured in samples per meter. In its normalized form, it is known as sample density (SD). A time-series for arbitrary or varying speeds is then obtained by interpolating the coefficients in a post processing step.

$$f_S = \frac{f_T}{\max|v|} \geq \frac{4}{\lambda_c} \tag{4}$$

$$\text{SD} = \frac{f_S}{\lambda_c/2} \geq 2 \tag{5}$$

3.1 Spatially Consistent Channel Parameters

GSCMs consist of two main components: a stochastic part that generates a random propagation environment, and a deterministic part that lets transmitters and receivers interact with this environment. In order to make realistic predictions of the wireless system performance, the random environment must fulfill certain statistical properties which are determined by measurements. These properties are captured by the so-called LSF model. A subsequent SSF model then generates individual scattering clusters for each MT. The joint spatial correlation of the model parameters as well as the correlated positions of the scattering clusters result in similarities in the communication channels of closely spaced MTs, *i.e.*, two co-located terminals must observe the same channel. Since the MT positions are given in 3-D Cartesian coordinates, all stochastic processes that determine the propagation environment need to be defined for 3-D coordinates as well. Many existing one-dimensional (1-D) correlations models (*e.g.*, [44], [45]), which are either a function of time or distance, must be extended to 3-D coordinates in order to be useful for GSCMs.

LSPs are more or less constant within an area of several meters. An example for this is the shadow fading (SF) which is caused by buildings or trees blocking a significant part of the signal. The so-called decorrelation distance of the SF, *i.e.*, the distance a MT must move to experience a significant change in the SF, is in the same order of magnitude as the size of the objects causing it. Thus, if a MT travels along a trajectory or if multiple MTs are closely spaced together, their LSPs are correlated. A common approach to model such correlation is by filtered Gaussian-distributed random numbers [46]. However, when it comes to spatial consistency, the positions of the scattering clusters must also be spatially correlated. The 3GPP proposal suggests that "spatially consistent powers/delays/angles of clusters are generated" [10]. However, this requires that all stochastic processes that determine the location of the scattering clusters are correlated. For a moderate scenario with 12 clusters and 20 sub-paths per cluster, this results in 2288 random variables[4]. Compared to the 7 variables needed for the LSF model, the filtering approach therefore requires prohibitively large amounts of memory and computing time.

Another problem arises when incorporating so-called *vertical industries* into the fifth generation (5G) infrastructure which is not yet covered by the 3GPP new-radio model but discussed in several ongoing research projects and standardization activities. Such *verticals* could be vehicular networks, air-to-ground communications, industrial peer-to-peer (P2P) communications, or communication scenarios involving satellites in low-earth orbit. All of these examples have in common that both ends of the link are mobile. However, the simultaneous mobility of both communication partners is not supported by the classical cellular shadowing models. Therefore, an alternative method for the generation of the random propagation environment is needed [47].

A computational efficient method to generate correlated stochastic processes has been introduced by Pätzold el. al. [48] who approximated the filtered white Gaussian noise process by a finite sum of properly weighted

[4]In the 3GPP new-radio model, scattering is based on the LOS / NLOS state (1 variable); LSPs (7 variables: delay spread, shadow fading, 4 angular spreads, K-factor); delays, powers, per-cluster angles ($10 \cdot 12$ variables); random coupling of sub-paths ($4 \cdot 12 \cdot 20$ variables); Cross polarization power ratios ($12 \cdot 20$ variables); Initial random phases ($4 \cdot 12 \cdot 20$ variables)

sinusoids. This idea was further developed into a 2-D shadowing model [11]. Wang et. al. then extended this 2-D method into a four-dimensional (4-D) method for device-to-device (D2D) channels [49]. A common problem for all these methods is finding the coefficients that best approximate a desired autocorrelation function (ACF) with a limited number of sinusoids.

In this section, a new approximation method is presented that allows the efficient calculation of the sinusoid coefficients for an arbitrary ACF in 3-D space[5]. It is shown how this translates directly into a six-dimensional stochastic process for D2D channels where both ends can be in different propagation environments, such as in air-to-ground channels.

3.1.1 The Sum of Sinusoids Model

A spatially correlated Gaussian stochastic process generates normal distributed random numbers

$$k(x,y,z) \sim \mathcal{N}(0,1), \tag{6}$$

with zero mean and unit variance. The value k is a function of the terminal location in 3-D Cartesian coordinates (x,y,z). From these numbers, other types of distributions can be obtained, *e.g.*, uniform or log-normal distribution with a different mean or variance. The 1-D spatial autocorrelation function (ACF) describes how fast the local mean of $k(x,y,z)$ evolves as a terminal moves. The ACF is usually modeled as an exponential decay function

$$\rho(d) = \exp\left(-\frac{d}{d_\lambda}\right), \tag{7}$$

with d as the distance between two positions and d_λ as the so-called decorrelation distance, *i.e.*, the distance at which the correlation between two samples falls below $e^{-1} \approx 0.37$ [44]. However, other types of decay functions may be desirable. The sum-of-sinusoids (SOS) method outlined in [48] approximates an 1-D Gaussian stochastic process $k(x)$ as a function of the position x on an 1-D linear trajectory as

$$\hat{k}(x) = \sum_{n=1}^{N} a_n \cos\left\{2\pi f_n x + \psi_n\right\} \tag{8}$$

with N sinusoids. The variables a_n, f_n, and ψ_n denote the amplitude, the frequency, and the phase of a sinusoid, respectively. The amplitudes a_n and the frequencies f_n are determined in a way that $\hat{k}(x)$ has similar statistical properties as $k(x)$, *i.e.*, $\hat{k}(x)$ has the same approximate ACF and the cumulative distribution function (CDF) is close to Gaussian density if N is sufficiently large. According to [48], 6 to 30 sinusoids are sufficient for an 1-D approximation. The phases ψ_n are randomly initialized in the range from $-\pi$ to π. Hence, exchanging the ψ_n while keeping a_n and f_n fixed creates a new spatially correlated stochastic process at minimal computational cost. A straight-forward expansion to a 3-D Gaussian stochastic process follows from [49] as

$$\hat{k}(x,y,z) = \sum_{n=1}^{N} a_n \cos\left\{2\pi\left(f_{x,n}x + f_{y,n}y + f_{z,n}z\right) + \psi_n\right\}. \tag{9}$$

Under the assumption that the fluctuations of $k(x,y,z)$ are wide sense stationary, the ACF only depends on the distance between two terminal positions. Hence, the 3-D spatial ACF of $\hat{k}(x,y,z)$ can be expressed as [49]

$$\hat{\rho}(\Delta x, \Delta y, \Delta z) = \sum_{n=1}^{N} \frac{a_n^2}{2} \cos\left\{2\pi\left(f_{x,n}\Delta x + f_{y,n}\Delta y + f_{z,n}\Delta z\right)\right\}. \tag{10}$$

When a MT moves from one location to another, the correlation $\rho(\Delta x, \Delta y, \Delta z)$ between the generated values $k(x,y,z)$ depends not only on the distance, but also on the direction of movement. In 3-D space,

[5] An extended version including a performance analysis was published in [12]

the traveling direction can be expressed by pitch and yaw. The pitch angle θ describes the vertical (tilt) angle relative to the horizontal plane. Positive rotation is up. The bearing or yaw angle ϕ describes the orientation on the ground. Here, it is defined in mathematic sense, *i.e.*, seen from above, a value of 0 points to the east and the angles increase counter-clockwise. In order to extend the existing 1-D approximations methods (*e.g.*, the L_p-norm method used in [45]) to 3 dimensions, we propose to assign a random direction (θ_n, ϕ_n) to each of the N sinusoids. There is no direct relationship between the direction (θ, ϕ) in which the MT moves and the directions (θ_n, ϕ_n) of the sinusoids. However, the sinusoid directions have to be chosen such that all possible movement directions of the MT are sufficiently covered, *e.g.*, by generating equidistributed points on the surface of a sphere as described in [50]. The resulting 3-D approximation of the Gaussian stochastic process (9) is then independent of the MT movement direction.

The directivity of the sinusoid frequencies is expressed as

$$f_{x,n}\Delta x = f_n \cdot d \cdot \cos\phi_n \cdot \cos\theta_n, \tag{11}$$

$$f_{y,n}\Delta y = f_n \cdot d \cdot \sin\phi_n \cdot \cos\theta_n, \tag{12}$$

$$f_{z,n}\Delta z = f_n \cdot d \cdot \sin\theta_n, \tag{13}$$

where f_n is the n-th root-frequency and d is the distance. Pätzold et. al. [48] proposed four methods to determine the amplitudes a_n and frequencies f_n in a SOS model. All of them show different performance in terms of average squared error (ASE) vs. the number of sinusoids, and computational complexity. They have in common that the approximated ACF is an exponential decay function (7). However, this is not always desirable since other types of decay functions might be needed. Hence, in the next section, a numeric approximation method is presented that calculates the sinusoid coefficients for arbitrary ACFs.

3.1.2 3-D Approximation of Arbitrary ACFs

The approximation method is derived from the Monte Carlo method [48, 51]. The sinusoid frequencies f_n are generated by an iterative optimization method that minimizes the error between the desired ACF and the approximate ACF for a given number of sinusoids. This method requires that the ACF is discretely sampled at $s = 1 \dots S$ sampling distances. This is done by defining a vector $\mathbf{d}$ that contains the sampling distances in increasing order.

$$\mathbf{d} = \begin{pmatrix} d_1 & d_2 & \dots & d_S \end{pmatrix}^T \tag{14}$$

Then, the sampled ACF $\rho(\mathbf{d})$ is obtained. The first distance value d_1 must be 0 and the first correlation value ρ_1 must be 1, *i.e.*, at zero-distance the generated values $\hat{k}(x, y, z)$ are identical. The N root-frequencies are randomly initialized to

$$f_n \sim \frac{1}{d_S} \cdot \mathcal{U}(-\pi, \pi), \tag{15}$$

where $\mathcal{U}(-\pi, \pi)$ describes an uniform distribution with values between $-\pi$ and π, and d_S the maximum distance for which the ACF is defined. The directional components $f_{x,n}$, $f_{y,n}$, and $f_{z,n}$ are calculated according to (11), (12), and (13) with $d = \Delta x = \Delta y = \Delta z = 1$, respectively.

The iterative optimization is done by updating the n-th root frequency while keeping all other $N - 1$ frequencies fixed. Then, the ASE is calculated for the overall 3-D space. Cai and Giannakis [11] introduced the ASE as a performance measure of the approximation. It is defined as the average squared error between the desired ACF $\rho(\mathbf{d})$ and the approximate ACF $\hat{\rho}(\Delta x, \Delta y, \Delta z)$. Here it is calculated as

$$\text{ASE} = \frac{1}{ST} \sum_{t=1}^{T} \sum_{s=1}^{S} \left\{ \rho(d_s) - \frac{1}{N} \sum_{n=1}^{N} \cos\left(\frac{2\pi}{d_S} f_{t,n} d_s \right) \right\}^2 . \tag{16}$$

Since the approximate ACF depends on the direction, the ASE calculation must take the directivity into account. Hence, the evaluation is done for $t = 1 \dots T$ randomly chosen *test* directions (θ_t, ϕ_t). The corresponding *test* frequencies $f_{t,n}$ are

$$f_{t,n} = (f_{x,n} \cos\phi_t + f_{y,n} \sin\phi_t) \cos\theta_t + f_{z,n} \sin\theta_t. \tag{17}$$

The ASE is used as a cost function for the *iterative refinement* of the sinusoid frequencies. If the ASE improves for a newly estimated root-frequency, the three sinusoid frequencies are replaced by the newly estimated ones, otherwise the new values are discarded. The iteration stops when no further improvement can be achieved for any of the N sinusoid components.

A new root-frequency is obtained by solving

$$f_n = \frac{1}{d_S} \arg\min_f \sum_{s=1}^{S} \left\{ \rho(d_s) - \hat{\rho}(d_s) - \frac{1}{N} \cos\left(\frac{2\pi}{d_S} f d_s \right) \right\}^2, \tag{18}$$

where $\rho(d_s)$ is the desired ACF and $\hat{\rho}(d_s)$ is the approximate ACF constructed from all $N-1$ components from the previous iteration. The amplitudes of all sinusoids are set to $a_n^2 = \frac{2}{N}$ and standard numerical methods can be applied to find the values f. However, (18) can only be used to approximate an 1-D stochastic process such as (8). For a 3-D stochastic process (9) it is necessary to obtain the three sinusoid components $f_{x,n}$, $f_{y,n}$, and $f_{z,n}$. This can be done by performing the estimation along one of the three axes of the coordinate system. The estimation direction (*i.e.*, the coordinate system axis) is chosen according to

$$\hat{\rho}(d_s) = \begin{cases} \hat{\rho}(\Delta x), & \text{for } \Delta x \geq \Delta y \text{ and } \Delta x \geq \Delta z; \\ \hat{\rho}(\Delta y), & \text{for } \Delta y > \Delta x \text{ and } \Delta y \geq \Delta z; \\ \hat{\rho}(\Delta z), & \text{for } \Delta z > \Delta x \text{ and } \Delta z > \Delta y, \end{cases} \tag{19}$$

where Δx, Δy, and Δz are calculated according to (11), (12), and (13) with $d = 1$, respectively. Then, one of the following calculation options is used:

Estimation in x-direction is performed by combining (11) and (10) while setting $\Delta y = \Delta z = 0$. This leads to a directional ACFs in x-direction

$$\hat{\rho}(\Delta x) = \frac{1}{N} \sum_{n=1}^{N} \cos(2\pi \cdot \Delta x \cdot \underbrace{f_n \cos\phi_n \cos\theta_n}_{=f_{x,n}}). \tag{20}$$

This function is sampled at the sampling distances (14) and used instead of $\hat{\rho}(d_s)$ in (18) to get an update of the n-th root frequency f_n. Due to the linear dependency, $f_{x,n}$, $f_{y,n}$, and $f_{z,n}$ are calculated from (11), (12), and (13).

Estimation in y-direction is performed by combining (12) and (10) while setting $\Delta x = \Delta z = 0$. This leads to a directional ACFs in y-direction

$$\hat{\rho}(\Delta y) = \frac{1}{N} \sum_{n=1}^{N} \cos(2\pi \cdot \Delta y \cdot \underbrace{f_n \sin\phi_n \cos\theta_n}_{=f_{y,n}}). \tag{21}$$

The n-th root frequency f_n is obtained and the directional components are calculated from (11)-(13).

Estimation in z-direction is performed by combining (13) and (10) while setting $\Delta x = \Delta y = 0$. This leads to a directional ACFs in z-direction

$$\hat{\rho}(\Delta z) = \frac{1}{N} \sum_{n=1}^{N} \cos(2\pi \cdot \Delta z \cdot \underbrace{f_n \sin\theta_n}_{=f_{z,n}}). \tag{22}$$

The n-th root frequency f_n is obtained and the directional components are calculated from (11)-(13).

The behavior of the approximated ACF is controlled in the overall 3-D space by calculating the ASE (16) for a large number of test directions and updating the directional sinusoid frequencies only if there is an overall improvement. The output of the approximation method are the sinusoid frequencies that can be used in (9) to generate spatially correlated normal-distributed random numbers with an arbitrary ACF and for arbitrary movements of the MT. It is possible to adjust the decorrelation distance d_λ and the distribution function of the stochastic process without having to recalculate the sinusoid frequencies. Doubling the distances in (14) is equal to halving the frequencies in (9). For example, if the approximation was done for $d_\lambda = 10$ m, but a stochastic process is needed for 20 m decorrelation distance, then simply dividing the frequencies by 2 creates the correct results. Uniform distribution can be achieved by a transformation from normal to uniform samples[6]. In the next section, it is shown how the SOS method can be used to generate correlated stochastic processes for P2P links where both ends of the communication channel are mobile.

3.1.3 Device-to-Device Extension

It was found in [49] that the MT movement at each end of the P2P link has an independent and equal effect on the correlation coefficient and that the joint correlation function (JCF) can be decomposed into two independent ACFs

$$\rho(\Delta x_t, \Delta y_t, \Delta z_t, \Delta x_r, \Delta y_r, \Delta z_r) = \rho(\Delta x_t, \Delta y_t, \Delta z_t) \cdot \rho(\Delta x_r, \Delta y_r, \Delta z_r). \tag{23}$$

The locations of the transmitting and receiving terminal are given in 3-D Cartesian coordinates as (x_t, y_t, z_t) and (x_r, y_r, z_r), respectively. [49] proposes to approximate a combined ACF. However, with six dimensions this results in a prohibitively large number of sinusoid coefficients and computing time. A simpler approach is to combine two independent Gaussian stochastic processes, one for the transmitter and one for the receiver into

$$k(x_t, y_t, z_t, x_r, y_r, z_r) = \frac{k_t(x_t, y_t, z_t) + k_r(x_r, y_r, z_r)}{\sqrt{2}}. \tag{24}$$

The approximated six-dimensional Gaussian process is

$$\hat{k}(x_t, y_t, z_t, x_r, y_r, z_r) = \sum_{n=1}^{2N} \frac{a_n}{\sqrt{2}} \cos\left\{2\pi \cdot \mathbf{f}_n^T \cdot [x_t \;\; y_t \;\; z_t \;\; x_r \;\; y_r \;\; z_r]^T + \psi_n\right\}, \tag{25}$$

where the sinusoid frequencies are obtained independently for the transmitting and receiving side.

$$\mathbf{f}_n = \begin{cases} \begin{bmatrix} f_{x_t,n} & f_{y_t,n} & f_{z_t,n} & 0 & 0 & 0 \end{bmatrix}^T, & \text{for } n \leq N; \\ \begin{bmatrix} 0 & 0 & 0 & f_{x_r,n} & f_{y_r,n} & f_{z_r,n} \end{bmatrix}^T, & \text{for } n > N. \end{cases} \tag{26}$$

The scaling by $\sqrt{2}$ in (24) accounts for doubling the number of SOS components in (25).

3.2 Large-Scale Fading Model

LSPs are relatively constant for several meters. An example is the SF which is caused by buildings or trees blocking a significant part of the signal. The so-called decorrelation distance of the SF, *i.e.*, the distance a MT must move to experience a significant change in the SF, is in the same order of magnitude as the size of the objects causing it. Thus, if a MT travels along a trajectory or if multiple MTs are closely spaced together, their LSPs are correlated. The positions and the reflective properties of the scattering clusters are based on eight LSPs:

1. RMS delay spread (DS)

[6]Given a standard Gaussian stochastic process $k \sim \mathcal{N}(0, 1)$, then remapping the probability density to $u \sim \mathcal{U}(0, 1)$ is done using the complementary error function as $u = 0.5 \cdot \text{erfc}(-k/\sqrt{2})$.

2. Ricean K-factor (KF)
3. Shadow fading (SF)
4. Azimuth spread of departure (ASD)
5. Azimuth spread of arrival (ASA)
6. Elevation spread of departure (ESD)
7. Elevation spread of arrival (ESA)
8. Cross-polarization ratio (XPR)

Their distribution properties are directly obtained from measurement data (*e.g.*, [4, 6–8, 52]). The granularity of each LSP can be described on three levels: the propagation scenario level, the link level, and the path level.

- **Propagation scenario level**
  The magnitude, variance and the correlation of a LSP in a specific scenario, *e.g.*, urban-macrocell, indoor hotspot, or urban satellite, are usually calculated from measurement data. Normally, LSPs are assumed to be log-normal distributed [53]. For example, the median log-normal delay spread DS_μ in an urban cellular scenario is $-6.89 \ \log_{10}(\text{s})$ which corresponds to a DS of $\sigma_\tau = 128$ ns[7]. With a standard deviation of $\text{DS}_\sigma = 0.5$, typical values may vary in between 40 and 407 ns. The same principle applies for the other six LSPs. The decorrelation distance (*e.g.*, $\text{DS}_\lambda = 40$ m) describes the distance-dependent correlation of the LSP. If *e.g.*, two mobile terminals in the above example are 40 m apart from each other, their DS is correlated with a correlation coefficient of $e^{-1} = 0.37$. Additionally, all LSPs are cross correlated. A typical example is the dependence of the AS, *e.g.*, the azimuth spread of arrival on the KF. With a large KF (*e.g.*, 10 dB), a significant amount of energy comes from a single direction. Thus, the AS gets smaller which leads to a negative correlation between the AS and the KF.

- **Link level**
  A MT at a specific position (black dot on the map in Figure 7) is assigned to a propagation scenario. Depending on the position and the scenario, it experiences a radio channel which is determined by the specific values of the seven LSPs. Due to the autocorrelation properties, small distances between users in the same scenario also lead to high correlations in the channel statistics, *e.g.*, a second terminal right next to the current user will experience a similar DS. The second granularity level thus contains the specific values of the LSPs for each user position. Generating those values can be seen as going from the scenario-wide distribution μ, σ of a LSP to virtual "*measurement*"-values for each MT.

- **Path Level**
  Finally, the individual components of the CIR are calculated. This procedure takes the values of the LSPs into account and calculates the path-powers and the path-delays of the MPCs. The detailed procedure for this is described in the following sections.

The communication scenario consists of multiple transmitters (TXs) and multiple receivers (RXs) operating at one ore more carrier frequencies simultaneously. Their locations are given in 3-D metric Cartesian coordinates as (x_t, y_t, z_t) and (x_r, y_r, z_r), respectively. The formulation of the LSF model is done for the dual-mobility case, where both ends of the link can be mobile. For any two channel realizations, the TX can be in a different position with d_t describing the distance between the two positions. Likewise, the RX can be displaced by a distance d_r. Single-mobility is a special case where either d_t or d_r is zero. The generation of properly correlated LSPs is then done in tree steps:

1. Generation of spatially correlated random variables in the logarithmic domain
2. Application of the inter-parameter correlations in the logarithmic domain
3. Transformation to the linear domain

[7]The model parameters are given in logarithmic units. To clearly indicate this, the units are defined in logarithmic scale as well. $\log_{10}(\text{s})$ is the logarithm of seconds and $\log_{10}(^\circ)$ is the logarithm of degrees.

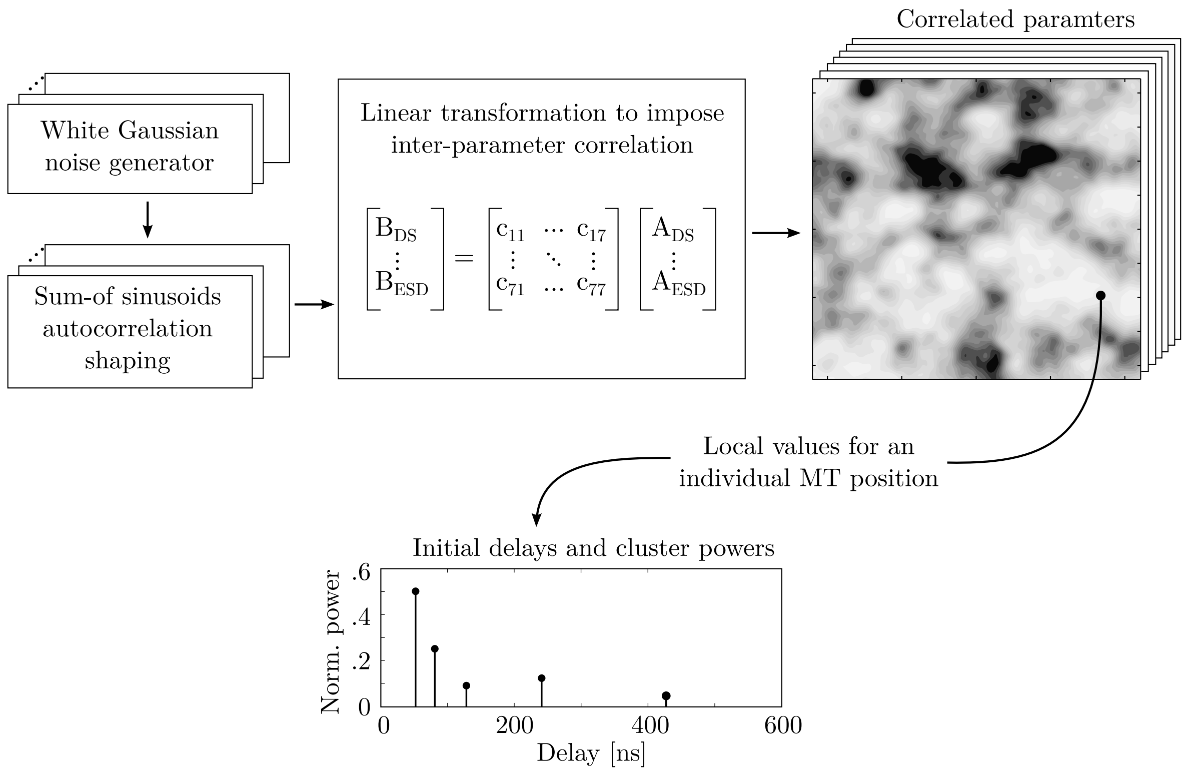

White Gaussian noise generator
Sum-of sinusoids autocorrelation shaping
Linear transformation to impose inter-parameter correlation
$\begin{bmatrix} B_{DS} \\ \vdots \\ B_{ESD} \end{bmatrix} = \begin{bmatrix} c_{11} & \cdots & c_{17} \\ \vdots & \ddots & \vdots \\ c_{71} & \cdots & c_{77} \end{bmatrix} \begin{bmatrix} A_{DS} \\ \vdots \\ A_{ESD} \end{bmatrix}$
Correlated paramters
Local values for an individual MT position
Initial delays and cluster powers
Norm. power
Delay [ns]

Figure 7: Principle of the generation of channel coefficients based on correlated LSPs

Spatial Correlation LSF Model The LSPs depends on up to four variables:

- The carrier frequency in GHz (f_{GHz})
- The 2-D distance between TX and RX on the ground in meters (d_{2D})
- The height of the TX above the ground in meters (h_{BS})
- The elevation angle (seen from the RX) between the TX and the ground in radians (α_{BS})

Based on these four variables, there are in between 3 and 12 parameters specified in the parameter tables. These 12 parameters are defined as follows:

1. The (mandatory) reference value μ (*mu*) at a carrier frequency of $f_{\text{GHz}} = 1$ GHz, a 2-D distance of $d_{\text{2D}} = 1$ m, a TX height of $h_{\text{BS}} = 1$ m and a TX elevation of $\alpha = 1$ rad or 57.3°
2. The (mandatory) reference standard deviation (STD) σ (*sigma*) at a carrier frequency of $f_{\text{GHz}} = 1$ GHz, a 2-D distance of $d_{\text{2D}} = 1$ m, a TX height of $h_{\text{BS}} = 1$ m and a TX elevation of $\alpha = 1$ rad or 57.3°
3. The (mandatory) decorrelation distance λ (*lambda*) in meters
4. The (optional) reference frequency offset ω (*omega*) in GHz
5. The (optional) frequency-dependence γ (*gamma*) of the reference value scaling with $\log_{10}(f_{\text{GHz}})$
6. The (optional) distance-dependence ϵ (*epsilon*) of the reference value scaling with $\log_{10}(d_{\text{2D}})$
7. The (optional) height-dependence ζ (*zeta*) of the reference value scaling with $\log_{10}(h_{\text{BS}})$
8. The (optional) elevation-dependence α (*alpha*) of the reference value scaling with $\log_{10}(\alpha_{\text{BS}})$
9. The (optional) frequency-dependence δ (*delta*) of the reference STD scaling with $\log_{10}(f_{\text{GHz}})$
10. The (optional) distance-dependence κ (*kappa*) of the reference STD scaling with $\log_{10}(d_{\text{2D}})$
11. The (optional) height-dependence τ (*tau*) of the reference STD scaling with $\log_{10}(h_{\text{BS}})$
12. The (optional) elevation-dependence β (*beta*) of the reference STD scaling with $\log_{10}(\alpha_{\text{BS}})$

The first three parameters, i.e., the reference value μ, the reference STD σ and the decorrelation distance λ must be specified for each LSP with exception of the SF, where the reference value μ is given by the path gain (PG). Parameters that are not specified are set to zero. Given the positions of the TX and the RX as $\mathbf{p} = (x_t, y_t, z_t, x_r, y_r, z_r)$, the initial value of the delay spread follows from

$$\begin{aligned}\mathrm{DS}_f(\mathbf{p}) \;=\; & \mathrm{DS}_\mu + \mathrm{DS}_\gamma \log_{10}(\omega + f_\mathrm{GHz}) + \mathrm{DS}_\epsilon \log_{10}(d_\mathrm{2D}) + \mathrm{DS}_\zeta \log_{10}(h_\mathrm{BS}) + \mathrm{DS}_\alpha \log_{10}(\alpha_\mathrm{BS}) + \\ & X^\mathrm{DS}(\mathbf{p}) \left\{ \mathrm{DS}_\sigma + \mathrm{DS}_\delta \cdot \log_{10}(\omega + f_\mathrm{GHz}) + \mathrm{DS}_\kappa \log_{10}(d_\mathrm{2D}) + \mathrm{DS}_\tau \log_{10}(h_\mathrm{BS}) + \mathrm{DS}_\beta \log_{10}(\alpha_\mathrm{BS}) \right\}.\end{aligned} \tag{27}$$

The variables d_2D, h_BS and α_BS are derived from the position vector $\mathbf{p}$ and are thus included in the resulting delay spread value. However, the carrier frequency f_GHz is independent of the positions. Hence, for a given set of carrier frequencies, there is an equal amount of DS values for each position. This is indicated by the index f. The random values $X^\mathrm{DS} \sim \mathcal{N}(0,1)$ is a spatially correlated Normal distributed random variables having zero-mean, unit variance and an ACF which is a combination of a Gaussian and an exponential ACF

$$\rho(d) = \begin{cases} \exp\left(-\frac{d^2}{d_\lambda^2}\right), & \text{for } d < d_\lambda; \\ \exp\left(-\frac{d}{d_\lambda}\right), & \text{for } d \geq d_\lambda, \end{cases} \tag{28}$$

where d_λ is the decorrelation distance defined by DS_λ in the parameter table. The variable d is the distance between two MTs positions. From observations in measurements it is known that, provided that TX and RX are in the same propagation environment, the DS is identical if TX and RX are swapped (this is known as channel reciprocity). Hence, the random variable X^DS is generated according to

$$\bar{X}^\mathrm{DS}(x_t, y_t, z_t, x_r, y_r, z_r) = \frac{\tilde{X}^\mathrm{DS}(x_t, y_t, z_t) + \tilde{X}^\mathrm{DS}(x_r, y_r, z_r)}{2 \cdot \sqrt{\rho_\tau(d_{tr}) + 1}}. \tag{29}$$

where $\tilde{X}^\mathrm{DS} \sim \mathcal{N}(0,1)$ is also a spatially correlated normal distributed random variable having zero mean and unit variance. However, $\tilde{X}$ depends only on three variables, i.e. the positions of the RX or TX. The scaling with $\sqrt{\rho(d_{tr}) + 1}$ ensures that the variance of the random process does not change for small TX-RX distances.

The same procedure applies to the KF and the XPR. The azimuth spread of departure (ASD), the azimuth spread of arrival (ASA), the ESD, the ESA use the same procedure for the generation of the reference values. However, a different approach is needed for the autocorrelation model since when TX and RX change places, the departure AS at the TX becomes the arrival AS at the RX and vice-versa. This effect can be captured by generating two independent 3-D spatially correlated random variables $\tilde{X}^{\mathrm{AS}d} \sim \mathcal{N}(0,1)$ and $\tilde{X}^{\mathrm{AS}a} \sim \mathcal{N}(0,1)$ and combining them to

$$\bar{X}^\mathrm{ASD}(x_t, y_t, z_t, x_r, y_r, z_r) = \frac{\tilde{X}^{\mathrm{AS}d}(x_t, y_t, z_t) + \tilde{X}^{\mathrm{AS}a}(x_r, y_r, z_r)}{2}, \tag{30}$$

$$\bar{X}^\mathrm{ASA}(x_t, y_t, z_t, x_r, y_r, z_r) = \frac{\tilde{X}^{\mathrm{AS}a}(x_t, y_t, z_t) + \tilde{X}^{\mathrm{AS}d}(x_r, y_r, z_r)}{2}. \tag{31}$$

The same procedure is repeated for the initial elevation angular spreads X^ESD and X^ESA. The reference value for the SF is the PG and hence, the SF does not require the values μ, γ, ϵ, ζ, and α.

Inter-Parameter Correlation Model In order to account for the inter-LSP correlation, a 8×8 matrix $\mathbf{R}$ is assembled containing all cross-correlation values ρ between each two LSPs.

$$\mathbf{R} = \begin{pmatrix} 1 & \rho_\mathrm{DS,KF} & \rho_\mathrm{DS,SF} & \rho_\mathrm{DS,ASD} & \rho_\mathrm{DS,ASA} & \rho_\mathrm{DS,ESD} & \rho_\mathrm{DS,ESA} & \rho_\mathrm{DS,XPR} \\ \rho_\mathrm{KF,DS} & 1 & \rho_\mathrm{KF,SF} & \rho_\mathrm{KF,ASD} & \rho_\mathrm{KF,ASA} & \rho_\mathrm{KF,ESD} & \rho_\mathrm{KF,ESA} & \rho_\mathrm{KF,XPR} \\ \rho_\mathrm{SF,DS} & \rho_\mathrm{SF,KF} & 1 & \rho_\mathrm{SF,ASD} & \rho_\mathrm{SF,ASA} & \rho_\mathrm{SF,ESD} & \rho_\mathrm{SF,ESA} & \rho_\mathrm{SF,XPR} \\ \rho_\mathrm{ASD,DS} & \rho_\mathrm{ASD,KF} & \rho_\mathrm{ASD,SF} & 1 & \rho_\mathrm{ASD,ASA} & \rho_\mathrm{ASD,ESD} & \rho_\mathrm{ASD,ESA} & \rho_\mathrm{ASD,XPR} \\ \rho_\mathrm{ASA,DS} & \rho_\mathrm{ASA,KF} & \rho_\mathrm{ASA,SF} & \rho_\mathrm{ASA,ASD} & 1 & \rho_\mathrm{ASA,ESD} & \rho_\mathrm{ASA,ESA} & \rho_\mathrm{ASA,XPR} \\ \rho_\mathrm{ESD,DS} & \rho_\mathrm{ESD,KF} & \rho_\mathrm{ESD,SF} & \rho_\mathrm{ESD,ASD} & \rho_\mathrm{ESD,ASA} & 1 & \rho_\mathrm{ESD,ESA} & \rho_\mathrm{ESD,XPR} \\ \rho_\mathrm{ESA,DS} & \rho_\mathrm{ESA,KF} & \rho_\mathrm{ESA,SF} & \rho_\mathrm{ESA,ASD} & \rho_\mathrm{ESA,ASA} & \rho_\mathrm{ESA,ESD} & 1 & \rho_\mathrm{ESA,XPR} \\ \rho_\mathrm{XPR,DS} & \rho_\mathrm{XPR,KF} & \rho_\mathrm{XPR,SF} & \rho_\mathrm{XPR,ASD} & \rho_\mathrm{XPR,ASA} & \rho_\mathrm{XPR,ESD} & \rho_\mathrm{XPR,ESA} & 1 \end{pmatrix} \tag{32}$$

Then, the square-root matrix $\mathbf{R}^{1/2}$ is calculated such that $\mathbf{R}^{1/2} \cdot \mathbf{R}^{1/2} = \mathbf{R}$ [54]. In order to calculate the matrix-square-root, $\mathbf{R}$ must be positive definite to get a unique, real-valued solution. The matrix $\mathbf{R}^{1/2}$ is multiplied with the eight normal distributed random variables

$$\begin{pmatrix} X^{\text{DS}}(\mathbf{p}) \\ X^{\text{KF}}(\mathbf{p}) \\ X^{\text{SF}}(\mathbf{p}) \\ X^{\text{ASD}}(\mathbf{p}) \\ X^{\text{ASA}}(\mathbf{p}) \\ X^{\text{ESD}}(\mathbf{p}) \\ X^{\text{ESA}}(\mathbf{p}) \\ X^{\text{XPR}}(\mathbf{p}) \end{pmatrix} = \mathbf{R}^{1/2} \cdot \begin{pmatrix} \bar{X}^{\text{DS}}(\mathbf{p}) \\ \bar{X}^{\text{KF}}(\mathbf{p}) \\ \bar{X}^{\text{SF}}(\mathbf{p}) \\ \bar{X}^{\text{ASD}}(\mathbf{p}) \\ \bar{X}^{\text{ASA}}(\mathbf{p}) \\ \bar{X}^{\text{ESD}}(\mathbf{p}) \\ \bar{X}^{\text{ESA}}(\mathbf{p}) \\ \bar{X}^{\text{XPR}}(\mathbf{p}) \end{pmatrix}, \tag{33}$$

where the variables $\bar{X}$ are uncorrelated and the variables X are cross-correlated. These cross-correlated variables are used in (27). In order to generate the correct statistical properties of the LSPs the parameters are generated in the following order:

1. Generate 3-D spatially correlated normal distributed random variables $\tilde{X}^{\text{DS}}$, $\tilde{X}^{\text{SF}}$, $\tilde{X}^{\text{KF}}$, $\tilde{X}^{\text{ASd}}$, $\tilde{X}^{\text{ASa}}$, $\tilde{X}^{\text{ESd}}$, $\tilde{X}^{\text{ESa}}$, and $\tilde{X}^{\text{XPR}}$ independently for each parameter by using the sum-of-sinusoids (SOS) model from section 3.1.1
2. Combine the 3-D variables $\tilde{X}$ to 6-D variables $\bar{X}$ by taking the TX and RX positions as well as channel reciprocity into account using (29) for the DS, SF, KF, and XPR; (30) for ASD and ESD; and (31) for ASA and ESA
3. Apply the inter-parameter correlation model (33) to obtain the 8 variables X
4. Calculate the variables $d_{2\text{D}}$, h_{BS} and α_{BS} from the TX and RX positions
5. Calculate the 8 initial LSPs using (27) for each TX-RX pair and for each carrier frequency

3.3 Multi-Frequency Small-Scale Fading Model

This section describes the small-scale-fading (SSF) model, i.e. the generation of individual multipath components for each MT. The 3GPP new-radio channel model introduced two additional features that, when implemented together, require some changes in the way the path parameters (delay, power, departure and arrival angles) are generated.

Firstly, 3GPP introduced spatial consistency, which solves one major drawback of previous GSCMs, namely the lack of realistic correlation in the SSF. Without spatial consistency, the positions of individual scattering clusters were generated randomly. With spatial consistency, the positions of the scattering clusters must be spatially correlated. One way to do this is by generating spatially consistent powers, delays and angles of the clusters which means that all random variables must be spatially correlated. An efficient way to do this is by utilizing the SOS method outlined in section 3.1. However, in order to be truly consistent, any function that modifies these random numbers must be continuous. For example, the delay generation in [10] requires to sort the delays ([10], eq. 7.5-2). Sorting is not a continuous function since it changes the order of the clusters depending on their delay. As a result, the channel coefficients show sudden "jumps" when plotting the phase over time on a continuous trajectory. The same happens for the scaling with the maximum path power when generating the arrival angles and departure angles ([10], eqs. 7.5-9 and 7.5-14), the positive or negative sign in the angles (eq. 7.5-16), and the random coupling of rays within a cluster. All these operations break the spatial consistency. The proposed SSF model will solve this issue by removing all non-continuous operations from the calculations.

The second new feature introduced by 3GPP are multi-frequency simulations. For this, an alternative channel generation method has been introduced as an optional modeling component. The idea is to generate random delays and angles from the given and delay and angular spreads at an anchor frequency. These delay and angles are the same for all frequencies. The cluster-powers are then generated independently for each frequency in the multi-frequency set by using scaling coefficients that take the different spreads at other frequencies into account. However, when the delay and angular spreads are calculated from the actual delays, powers and angles, they do not match the given spreads from the LSF model, which is generally not the case for the family of 3GPP models.

A third new feature, which is not yet part of the 3GPP new-radio model, is the support for P2P communications, such as in vehicular networks, air-to-ground communications, industrial communications, or communication scenarios involving satellites in low-earth orbit. All of these examples require that both ends of the link are mobile. Dual-mobility has already been incorporated into the LSF model by taking the positions of both the TX and the RX into account. The same approach can be used for the spatially consistent delays and angles. However, some additional modifications are needed for the case when TX and RX are close together.

In this section, we propose a modified version of the 3GPP alternative channel generation method that works in three steps: First, spatially consistent delays and angles are generated without taking the spreads from the LSF model into account. Second, the multi-frequency scaling is done for the powers in a similar way as suggested by 3GPP. Finally, the delay and angles are adjusted to match the given spreads. This ensures that the output of the SSF model is consistent with the LSF model.

3.3.1 Communication Model

The communication model consists of multiple transmitters (TXs) and multiple receivers (RXs). Their locations are given in 3-D metric Cartesian coordinates as (x_t, y_t, z_t) and (x_r, y_r, z_r), respectively. A transmitted signal is reflected and scattered by objects in the environment such that multiple copies of that signal are received by the RX. Each *path* that a signal takes consists of a departure direction at the TX, a first-bounce scatterer (FBS), a LBS, and an arrival direction at the RX. Departure and arrival directions are given in geographic coordinates consisting of an azimuth angle ϕ and an elevation angle θ[8]. The formulation of the path generation procedure is done for the dual-mobility case, where both ends of the link can be mobile. For any two channel realizations, the TX can be in a different position with d_t describing the distance between the two positions. Likewise, the RX can be displaced by a distance d_r. Single-mobility is a special case where either d_t or d_r is zero. All random variables that determine the positions of the NLOS scatterers are spatially correlated, i.e. they depend on the positions of the TX and the RX [12].

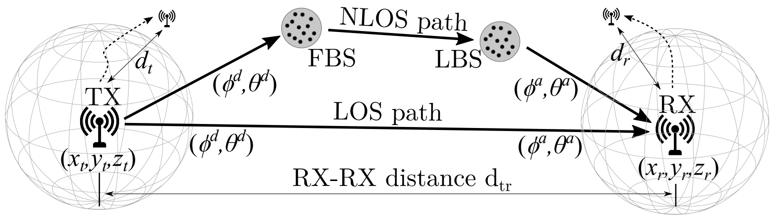


Figure 8: Illustration of the communication model

3.3.2 Initial Delays and Angles

Initial delays for the NLOS paths are drawn randomly from a single-sided exponential distribution with unit mean and unit STD as

$$\tilde{\tau}_l = -\ln\left\{X_l^\tau(x_t, y_t, z_t, x_r, y_r, z_r)\right\}, \tag{34}$$

[8] ϕ is defined in mathematic sense, i.e., seen from above, a value of 0 points to the east and the angles increase counter-clockwise. θ describes the rotation relative to the horizontal plane. Positive rotation is up.

where the index l denotes the path number and $X_l^\tau \sim \mathcal{U}(0,1)$ is a spatially correlated uniformly distributed random variable having values between 0 and 1. The LOS delay, i.e. the delay of the first path, is set to 0. The initial delays are not scaled by the DS nor are the angles scaled by the AS. The actual spread values will be applied later. This approach is different compared to the new radio (NR) model [10] which includes the spreads already in the initial values. The autocorrelation function (ACF) of X_l^τ is a combination of a Gaussian and an exponential ACF

$$\rho_\tau(d) = \begin{cases} \exp\left(-\frac{d^2}{d_\lambda^2}\right), & \text{for } d < d_\lambda; \\ \exp\left(-\frac{d}{d_\lambda}\right), & \text{for } d \geq d_\lambda, \end{cases} \tag{35}$$

where d is the distance between two MTs positions and d_λ is the decorrelation distance, i.e. the distance at which the correlation falls below $e^{-1} \approx 0.37$ [44]. The combined ACF produces a higher correlation between delays of closely spaced MTs. It was found that this is more realistic than an exponential ACF, where the delays may vary significantly even when MTs move only a few centimeters. From observations in measurements it is known that the path delays are identical if TX and RX are swapped (this is known as channel reciprocity). Hence, the random variable X_l^τ is generated according to

$$X_l^\tau(x_t, y_t, z_t, x_r, y_r, z_r) = \frac{1}{2}\text{erfc}\left(-\frac{\tilde{X}_l^\tau(x_t, y_t, z_t) + \tilde{X}_l^\tau(x_r, y_r, z_r)}{2 \cdot \sqrt{\rho_\tau(d_{tr}) + 1}}\right), \tag{36}$$

where $\tilde{X}_l^\tau \sim \mathcal{N}(0,1)$ is a spatially correlated normal distributed random variable having zero mean and unit variance, and erfc is the complementary error function. The scaling with $\sqrt{\rho_\tau(d_{tr}) + 1}$ ensures that the variance of the random process does not change for small TX-RX distances.

Next, initial values for the four angles are generated for each path. As for the DS, they are generated spatially consistent having the same ACF and decorrelation distance. The NR model [10] proposes to obtain the azimuth angles from a wrapped Gaussian distribution, the elevation angles from a wrapped Laplacian distribution, and the path power from a single slope exponential power delay profile (PDP). However, this leads to large angle offsets when the powers are small and the AS is large. Due to the wrapping operation, the achievable AS is limited. In order to have a larger range of possible angular spreads, we propose to draw the initial NLOS angles from a uniform distribution as

$$\tilde{\phi}_l = X_l^\phi(x_t, y_t, z_t, x_r, y_r, z_r) \sim \mathcal{U}\left(-\frac{\pi}{2}, \frac{\pi}{2}\right). \tag{37}$$

As for the DS, the actual AS is later applied by a scaling operation. If TX and RX change places, the departure angles at the TX become the arrival angles at the RX and vice-versa. This effect can be captured by generating two independent 3-D spatially correlated random variables $\tilde{X}_l^{\phi d} \sim \mathcal{N}(0,1)$ and $\tilde{X}_l^{\phi a} \sim \mathcal{N}(0,1)$ and combining them to

$$\tilde{\phi}_l^d(x_t, y_t, z_t, x_r, y_r, z_r) = \frac{\pi}{2}\text{erfc}\left(-\frac{\tilde{X}_l^{\phi d}(x_t, y_t, z_t) + \tilde{X}_l^{\phi a}(x_r, y_r, z_r)}{2}\right) - \frac{\pi}{2}, \tag{38}$$

$$\tilde{\phi}_l^a(x_t, y_t, z_t, x_r, y_r, z_r) = \frac{\pi}{2}\text{erfc}\left(-\frac{\tilde{X}_l^{\phi a}(x_t, y_t, z_t) + \tilde{X}_l^{\phi d}(x_r, y_r, z_r)}{2}\right) - \frac{\pi}{2}. \tag{39}$$

The same procedure is repeated for the initial elevation angles $\tilde{\theta}_l^d$ and $\tilde{\theta}_l^a$. The initial angles for the LOS path are set to 0.

3.3.3 Initial Path Powers

The initial delays $\tilde{\tau}_l$ and the four initial angles $\tilde{\phi}_l^d$, $\tilde{\phi}_l^a$, $\tilde{\theta}_l^d$, and $\tilde{\theta}_l^a$ are assumed to be frequency-independent, i.e. a MT sees the same scattering clusters at different frequencies and therefore, the same angles and delays are used. However, DS and AS are generally frequency-dependant. For example, the DS at a carrier frequency of 10 GHz is generally shorter than at 2 GHz. The NR model [10] proposes an alternative channel generation method which can adjust the path powers such that different delay and angular spreads can be achieved. The powers are calculated as

$$\tilde{P}_{l,f} = \exp\left\{-\tilde{\tau}_l \cdot g_f^{DS} - \left(\tilde{\phi}_l^d\right)^2 \cdot g_f^{ASD} - \left(\tilde{\phi}_l^a\right)^2 \cdot g_f^{ASA} - \left|\tilde{\theta}_l^d\right| \cdot g_f^{ESD} - \left|\tilde{\theta}_l^a\right| \cdot g_f^{ESA}\right\}, \tag{40}$$

where the index f refers to the $f = 1 \dots F$ carrier frequencies. The scaling coefficients g_f^{DS}, g_f^{ASD}, g_f^{ASA}, g_f^{ESD}, and g_f^{ESA} need to be calculated such that the frequency-dependent differences in the spreads are reflected in the powers. Given that the initial delays (34) are drawn from a single-sided exponential distribution with unit mean and unit STD, the maximum DS can be 1 if all paths have the same power. By choosing the path powers as $\tilde{P}_l = \exp\{-9 \cdot \tilde{\tau}_l\}$, a DS of is 0.1 is obtained. This can be seen as a lower bound. In other words, the smallest DS in a multi-frequency simulation must be at least 10% of the largest value[9]. The achievable DS values as a function of the scaling coefficient g are listed in Table 34. For single-frequency simulations, the NR model [10] uses the delay distribution proportionality factor r_τ to shape the PDP. r_τ typically has values in between 1.7 and 3.8 and it follows from ([10], eq. 7.5-5) that $g^{DS} = r_\tau - 1$. However, for the multi-frequency simulations, there might be a different DS for each frequency. In this case, g_f^{DS} is a function of the frequency and so is r_τ. Hence, it is not possible to fix r_τ to a given value. Instead, the DS gets mapped to g_f^{DS} by

$$g_f^{DS} = -1.5 \cdot \ln\left\{1.2 \cdot \overline{\text{DS}}_f - 0.15\right\}, \text{ with } \quad \overline{\text{DS}}_f = \frac{\text{DS}_f}{\max(\text{DS}_f) + \min(\text{DS}_f)}, \quad 0.15 < \overline{\text{DS}}_f < 0.85. \tag{41}$$

The parameters of this mapping function were numerically obtained by fitting the values from Table 34 to an exponential function. If there is no frequency dependency of the DS, then g_f^{DS} will have a value of 1.2 which corresponds to $r_\tau = 2.2$. A similar mapping is done for the AS. The initial angles are uniformly distributed having values in between $-\pi/2$ and $\pi/2$. For the azimuth angles, the path powers are mapped to a Gaussian PAS. As for the DS, the scaling coefficients g_f^{ASD} and g_f^{ASA} must be obtained. The achievable AS for different scaling coefficients are listed in Table 34 and the mapping is done by

$$g_f^{ASD/A} = -2.2 \cdot \ln\left\{1.5 \cdot \overline{\text{AS}}_f - 0.35\right\}, \text{ with } \quad \overline{\text{AS}}_f = 0.75 \cdot \frac{\text{AS}_f}{\max(\text{AS}_f)}, \quad \overline{\text{AS}}_f > 0.25. \tag{42}$$

If there is no frequency dependency of the azimuth AS, then $g_f^{ASD/A}$ will have a value of 0.56 which corresponds to an initial spread of 42°. This can be scaled down to 14° by adjusting the path powers. Lastly, the scaling coefficients g_f^{ESD} and g_f^{ESA} for the elevation angles are obtained. The path powers are mapped to a Laplacian PAS by

$$g_f^{ESD/A} = -3.4 \cdot \ln\left\{1.2 \cdot \overline{\text{ES}}_f - 0.1\right\}, \text{ with } \quad \overline{\text{ES}}_f = 0.75 \cdot \frac{\text{ES}_f}{\max(\text{ES}_f)}, \quad \overline{\text{ES}}_f > 0.25. \tag{43}$$

If there is no frequency dependency of the elevation AS, then $g_f^{ESD/A}$ will have a value of 0.76 which corresponds to an initial elevation spread of 44° which can be scaled down to 14° at a different frequency.

Table 34: Achievable delay and angular spread values

Scaling coeff. g	0.0	0.5	1.0	2.0	3.0	5.0	9.0
DS [s]	1.00	0.67	0.50	0.33	0.25	0.17	0.10
ASD/A [deg]	51.6	43.6	37.0	28.1	23.1	17.9	13.2
ESD/A [deg]	51.6	46.4	41.3	31.9	24.6	15.7	8.6

[9]The actual values are later applied by adjusting the path delays. Here, only the relative difference between maximum and minimum DS is of interest.

3.3.4 Applying K-Factor, Delay Spread and Angle Spreads

The initial delays $\tilde{\tau}_l$ and cluster powers $\tilde{P}_{l,f}$ are chosen such that the DS has values around 0.5 seconds. The cluster angles $\tilde{\phi}_l^d$, $\tilde{\phi}_l^a$, $\tilde{\theta}_l^d$, $\tilde{\theta}_l^a$ are initialized such that the initial ASs have values around 40°. Now, the actual values of the KF, the DS and the four ASs are applied. In multi-frequency simulations, the cluster delays and angles are independent of the carrier frequency, i.e. the same clusters are seen independent of the frequency. However, the cluster powers are frequency-dependent and reflect relative differences in the delay and angular spreads. For example, the DS might decrease if the carrier frequency increases from 2 GHz to 10 GHz. The scaling of the initial cluster powers in (40) already takes this relative difference into account by assigning less power to the clusters with a longer delay at 10 GHz.

Applying the K-Factor The KF is defined as the ratio of the power of the direct path divided by the sum-power of all other paths. As for the DS and the ASs, the KF is generally frequency-dependent. It is applied by setting the power of the first path to

$$\tilde{P}_{1,f} = \text{KF}_f \cdot \sum_{l=2}^{L} \tilde{P}_{l,f}. \tag{44}$$

Then, the path powers are normalized such that their sum power is one Watt. This is done independently for each frequency.

$$P_{l,f} = \tilde{P}_{l,f} / \sum_{l=1}^{L} \tilde{P}_{l,f} \tag{45}$$

Applying the Delay Spread The DS measures how the multipath power is spread out over time. Given the cluster-powers $P_{l,f}$ from (45) and the initial delays $\tilde{\tau}_l$ from (34), the initial DS is calculated as

$$\widetilde{\text{DS}}_f = \sqrt{\frac{1}{P_f} \cdot \sum_{l=1}^{L} P_{l,f} \cdot (\tilde{\tau}_l)^2 - \left(\frac{1}{P_f} \cdot \sum_{l=1}^{L} P_{l,f} \cdot \tilde{\tau}_l \right)^2}, \tag{46}$$

where P_f is the sum-power of all clusters at frequency f. The values of $\widetilde{\text{DS}}_f$ are frequency-dependent due to the scaling of the path-powers in (40), but do not contain the correct DS values from the LSF model. Hence, the delays need to be scaled such that the correct DS can be calculated from the generated path-delay and path-powers. This is done by

$$\tau_l = \tilde{\tau}_l \cdot \frac{1}{F} \cdot \sum_{f=1}^{F} \frac{\text{DS}_f}{\widetilde{\text{DS}}_f}. \tag{47}$$

The last step is different from other models. The WINNER [4] and 3GPP-3D model [24] scale the delays with an empiric formula that corrects the delays to reduce the effect of a high KF. However, due to the random variables in (34) the resulting DS is always different from the value in the LSF model. The proposed method ensures that scattering clusters are distributed in a way that the DS calculated from the MPCs is exactly the same as the DS from the LSF model. In the following section, the departure and arrival directions of each MPC are generated. Those are the combined with the delays in order to calculate the 3-D positions of the scattering clusters.

Applying the Angular Spreads The AS measures how the multipath power is spread out in the spatial domain. The AS is ambiguous since the angles are distributed on a sphere and the resulting value depends on the reference angle, *i.e.*, the definition of where 0° is. A linear shift of the angles $\phi_l + \Delta_\phi$ leads to the AS being a function of Δ_ϕ. In the 3GPP SCM [55], this was solved by an exhaustive search over $\Delta_\phi \in [-\pi, \pi[$ to find the minimum AS

$$\sigma_\phi = \min_{\Delta_\phi} \sigma_\phi(\Delta_\phi). \tag{48}$$

In a similar approach the angles are normalized such that the combined power-angular spectrum (PAS) of all paths and sub-paths points to $\theta = \phi = 0$. The normalization of the angle $\tilde{\phi}_l$ is done by

$$\Delta_{\phi_f} = \arg\left(\sum_{l=1}^{L} \exp\left\{j\tilde{\phi}_l\right\} \cdot P_{l,f}\right), \tag{49}$$

$$\hat{\phi}_{l,f} = \arg\exp\left\{j\left(\tilde{\phi}_l - \Delta_{\phi_f}\right)\right\}. \tag{50}$$

Then, the AS is calculated by

$$\widetilde{\text{AS}}_f = \sqrt{\frac{1}{P_f} \cdot \sum_{l=1}^{L} P_{l,f} \cdot \left(\hat{\phi}_{l,f}\right)^2 - \left(\frac{1}{P_f} \cdot \sum_{l=1}^{L} P_{l,f} \cdot \hat{\phi}_{l,f}\right)^2}. \tag{51}$$

With AS_f being the initial AS from the LSF model, the initial angles $\tilde{\phi}_l$ are scaled to

$$\phi_l = \arg\exp\left(j \cdot \tilde{\phi}_l \cdot s\right), \quad s = \frac{1}{F} \cdot \sum_{f=1}^{F} \frac{\text{AS}_f}{\widetilde{\text{AS}}_f}, \quad s < \begin{cases} 3.0, & \text{for azimuth angles;} \\ 1.5, & \text{for elevation angles.} \end{cases} \tag{52}$$

The arg exp function wraps the angles around the unit circle. The scaling coefficient s is limited to a maximum value of 3 for the scaling of the azimuth angles and 1.5 for the elevation angles. This is motivated by the distribution of the initial angles in (37). More power is assigned to the angles having values close to 0 than to those having values close to $\pm\frac{\pi}{2}$. For this reason, $s = 3$ achieves the maximum azimuth AS of around 85° and $s = 1.5$ achieves the maximum elevation AS of around 48°. Larger values of s tend to decrease the AS again due to the wrapping around the unit circle.

LOS angles The last step is to apply the direction of the LOS path. The initial values of the LOS angles $\tilde{\phi}_1^d$, $\tilde{\phi}_1^a$, $\tilde{\theta}_1^d$, and $\tilde{\theta}_1^a$ were set to 0. However, the correct angles need to take the positions of the TX and the RX into account. The LOS angles are

$$\phi_1^d = \arctan_2\left\{y_r - y_t, x_r - x_t\right\}, \tag{53}$$

$$\phi_1^a = \phi_1^d + \pi \tag{54}$$

$$\theta_1^d = \arctan_2\left\{z_r - z_t, d_{2d}\right\}, \tag{55}$$

$$\theta_1^a = -\theta_1^d, \tag{56}$$

$$d_{2d} = \sqrt{(x_r - x_t)^2 + (y_r - y_t)^2} \tag{57}$$

where $\arctan_2$ is the multi-valued inverse tangent. Those angles are applied by two 3-D rotations in Cartesian coordinates, one for the TX and one for the RX. The operations are identical. The NLOS departure and arrival angles from the precious calculation are converted to Cartesian coordinates by

$$\mathbf{c}_l = \begin{pmatrix} \cos\theta_l \cdot \cos\phi_l \\ \cos\theta_l \cdot \sin\phi_l \\ \sin\theta_l \end{pmatrix}. \tag{58}$$

Then, a rotation matrix is constructed from the LOS angles. This matrix is a combined rotation around the y-axis followed by a rotation around the z-axis in Cartesian coordinates. It is applied by

$$\hat{\mathbf{c}}_l = \begin{pmatrix} \cos\theta_1 \cdot \cos\phi_1 & -\sin\phi_1 & -\sin\theta_1 \cdot \cos\phi_1 \\ \cos\theta_1 \cdot \sin\phi_1 & \cos\phi_1 & -\sin\theta_1 \cdot \sin\phi_1 \\ \sin\theta_1 & 0 & \cos\theta_1 \end{pmatrix} \cdot \mathbf{c}_l. \tag{59}$$

The final angles are then obtained by converting $\hat{\mathbf{c}}_l$ back to spherical coordinates.

$$\phi_l = \arctan_2 \{\hat{c}_{l,x}, \hat{c}_{l,y}\} \quad (60)$$

$$\theta_l = \arctan_2 \left\{\hat{c}_{l,z}, \sqrt{\hat{c}_{l,x}^2 + \hat{c}_{l,y}^2}\right\} \quad (61)$$

3.3.5 Subpaths

At bandwidths below 100 MHz, closely clustered paths that originate from the same scattering cluster cannot be resolved in the delay domain. However, those unresolvable paths can still arrive from slightly different directions. When the band-limited CIR is observed over time, the power of the *resolved paths* appears to fluctuate as a result of the constructive and destructive interference caused by the superposition of the unresolvable paths. This phenomenon was modeled in the 3GPP-SCM [22] by splitting NLOS paths into 20 sub-paths that arrive from slightly different directions. The LOS path has no sub-paths. The same approach is used in the new model. The azimuth and elevation angles of each sub-paths are calculated as

$$\phi_{l,m} = \phi_l + \frac{\pi \cdot c_\phi \cdot \hat{\phi}_m}{180°}, \quad \text{for } l > 1; \quad (62)$$

$$\theta_{l,m} = \theta_l + \frac{\pi \cdot c_\theta \cdot \hat{\phi}_m}{180°}, \quad \text{for } l > 1. \quad (63)$$

Here, m is the sub-path index, c_ϕ is the scenario-dependent cluster-wise RMS AS and $\hat{\phi}$ is the offset angle of the mth sub-path from Table 35. c_ϕ and $\hat{\phi}$ are given in degrees here. Furthermore, each of the 20 angle pairs $({\theta^d}_{l,m}, {\phi^d}_{l,m})$ at the TX gets coupled with a random angle pair $({\theta^a}_{l,m}, {\phi^a}_{l,m})$ at the RX (see [4]).

Table 35: Offset Angle of the m^{th} Sub-Path from [4]

Sub-path m	Offset angle $\hat{\phi}_m$ (degrees)	Sub-path m	Offset angle $\hat{\phi}_m$ (degrees)
1,2	± 0.0447	11,12	± 0.6797
3,4	± 0.1413	13,14	± 0.8844
5,6	± 0.2492	15,16	± 1.1481
7,8	± 0.3715	17,18	± 1.5195
9,10	± 0.5129	19,20	± 2.1551

At this point, each (sub-)path is described by its delay, power, departure direction, and arrival direction. The next step is to use these values to calculate the exact position of the scatterers. This is needed in order to incorporate mobility at the MT. When the MT moves to a different location, the delays and arrival directions are updated in a deterministic way. One method for doing this was proposed by Baum *et al.* [26] who introduced the term *drifting*.

3.4 Drifting

After the path-delays, powers, and angles are known for the *initial positions* of the TX and RX, their values are updated when the MTs move to a different location. This is an essential step to ensure spatial consistency for moving terminals. Without tracking of the delays and directions of a path, the behavior of the channel coefficients at the output of the model does not agree well with the reality. State-of-the-art GSCMs such as the 3GPP-3D [9] and the new radio model [10] use a simplified approach for the temporal evolution of the CIR. In this approach, each MPC gets assigned a Doppler shift based on the initial arrival angles. However, the angles and and path delays remain unchanged. In these models, there is no way to explicitly describe the movements of a terminal, *e.g.*, pedestrians using mobile phones, people in trains, cars, or other means of transport. This often leads to simplified simulation assumptions such as all MTs having ideal antennas and antenna orientations, moving at a constant speed and in a fixed direction. Conclusions drawn from such simulations might be very different from the achievable performance in real deployments.

In the dual-mobility model, both TX and RX move along trajectories. The channel coefficients are calculated at so-called snapshot positions, i.e. at an ordered list of positions defining the trajectory. The dual-mobility model requires that the TX and RX trajectories have the same number of snapshot positions but do not need to have the same length. Hence, both sides of the communication link can move at different speeds such as in air-to-ground channels. Single-mobility is obtained when the TX trajectory has zero-length, i.e. the TX remains stationary while the RX is moving. Due to channel reciprocity, forward and return channels are identical and, therefore, this also covers the case when the TX is mobile and the RX is stationary. The RX trajectory is divided into segments. Each segment is several meters long and it is assumed that the channel maintains its wide sense stationary (WSS) properties for the time it takes a MT to traverse the segment, *i.e.*, the LSPs don't change significantly. The TX trajectory does not contain segments but it "inherits" the segments from the RX. For example, if a segment start at snapshot number 500 of the RX trajectory, it would also start at at snapshot number 500 of the TX trajectory.

Changes in the orientation of a terminals are treated in Section 3.5 when the antenna characteristics are included. The *birth and death* of MPCs is treated at the edge of a segment when a new segment starts (see Section 3.8). Here, a method to update the path-parameters (delay, angles, and phase) for each MT position along a segment is presented. Drifting for 2-D propagation was already introduced in an extension of the SCM [26]. However, it was not incorporated into the WINNER and 3GPP-3D models and no evaluation was reported. Here, the idea from [26] is extended towards 3-D propagation to incorporate time evolution into the new model.

The following paragraphs outline the calculations needed to implement drifting in 3-D coordinates which are not part of any of the previous GSCMs. For this, the individual delays and angles for each MPC from the previous sections are needed. In order to obtain correct results for large array antennas, *e.g.*, for massive MIMO, the calculations must be done for each individual element of an array antenna. Thus, the new model inherently supports spherical waves. If the BS array size is small compared to the BS-MT distance, it is sufficient to consider only a single scatterer (the LBS) for the NLOS paths. In this case, all calculations at the BS assume planar waves and the delays, angles and phases are updated for different MT positions with respect to the LBS. This is done in the so-called *single-bounce model*. However, many massive MIMO deployments are done indoors or in so-called micro-cell scenarios where BS and MT are relatively close to each other. At the same time, scattering clusters might be very close to the transmitter which is not the case in macro-cell scenarios when the BS antennas are deployed above the rooftop. For such micro-cell deployments it is essential to calculate the correct phase for each BS antenna element. Thus, in addition to the LBS, it is also necessary to take the position of the FBS into account. This is done in the *multi-bounce model*. Since the single-bounce model is a special case of the multi-bounce model (FBS and LBS are the same), both approaches are used. In the following, the single-bounce model is described first and then extended towards the multi-bounce model. The special LOS case is described last.

Besides the initial delays, path-powers, and angles, drifting requires the exact position of each antenna element. At the MTs, element positions need to be updated for each snapshot with respect to the MT orientation. The following calculations are then done element-wise. The indices denote the RX antenna element (r) and the TX antenna element (t), the path number (l), the sub-path number (m), and the snapshot number (s) within the current segment, respectively. The scatterer positions are kept fixed for the time it takes a MT to move through a segment.

NLOS drifting (single-bounce model) The position of the LBS is calculated based on the initial arrival angles and the path delays. Then, the angles and path lengths between the LBS and the terminals are updated for each snapshot on the trajectories. This is done for each antenna element separately. Figure 9 illustrates the angles and their relations. Since only the arrival angles are used for this calculation, initial departure angles are omitted. Since the initial delay of the LOS path is normalized to zero, the total length of the l^{th} path follows from

$$d_l = \tau_l \cdot c + |\mathbf{r}|\,, \tag{64}$$

where $|\mathbf{r}|$ is the distance between the initial TX and the initial RX location and c is the speed of light. All sub-paths have the same delay and thus the same path length. However, each sub-path has different arrival angles $(\theta^a_{l,m}, \phi^a_{l,m})$. Those angles are transformed into Cartesian coordinates to obtain

$$\hat{\mathbf{a}}_{l,m} = \begin{pmatrix} \cos\phi^a_{l,m} \cdot \cos\theta^a_{l,m} \\ \sin\phi^a_{l,m} \cdot \cos\theta^a_{l,m} \\ \sin\theta^a_{l,m} \end{pmatrix} = \frac{\mathbf{a}_{l,m}}{|\mathbf{a}_{l,m}|}. \tag{65}$$

This vector has unit length and points from the initial RX position towards the scatterer. Next, the length of the vector $\mathbf{a}_{l,m}$ is obtained. Since the drifting at the MT is modeled by a single reflection, TX, RX, and LBS form a triangle. The values of d_l, $\mathbf{r}$, and $\hat{\mathbf{a}}_{l,m}$ are known. Thus, the cosine theorem can be used to calculate the length $|\mathbf{a}_{l,m}|$ between the RX and LBS.

$$|\mathbf{b}_{l,m}|^2 = |\mathbf{r}|^2 + |\mathbf{a}_{l,m}|^2 - 2\,|\mathbf{r}|\,|\mathbf{a}_{l,m}|\cos\alpha_{l,m} \tag{66}$$

$$(d_l - |\mathbf{a}_{l,m}|)^2 = |\mathbf{r}|^2 + |\mathbf{a}_{l,m}|^2 + 2\,|\mathbf{a}_{l,m}|\,\mathbf{r}^T\hat{\mathbf{a}}_{l,m} \tag{67}$$

$$|\mathbf{a}_{l,m}| = \frac{d_l^2 - |\mathbf{r}|^2}{2\cdot(d_l + \mathbf{r}^T\hat{\mathbf{a}}_{l,m})} \tag{68}$$

In (67) the term $\cos\alpha_{l,m}$ is substituted with $-\mathbf{r}^T\hat{\mathbf{a}}_{l,m}/|\mathbf{r}|$ since the angle between $\mathbf{a}_{l,m}$ and $-\mathbf{r}$ is needed. Now, the vector $\mathbf{a}_{r,l,m,s}$ is calculated for the RX antenna element r at snapshot s. The element position includes the orientation of the array antenna with respect to the moving direction of the RX. Hence, the vector $\mathbf{e}_{r,s}$ points from the initial RX location to the r^{th} antenna element at snapshot s.

$$\mathbf{a}_{r,l,m,s} = \mathbf{a}_{l,m} - \mathbf{e}_{r,s} \tag{69}$$

An update of the arrival angles is obtained by transforming $\mathbf{a}_{r,l,m,s}$ back to spherical coordinates.

$$\phi^a_{r,l,m,s} = \arctan_2\{a_{r,l,m,s,y}, a_{r,l,m,s,x}\} \tag{70}$$

$$\theta^a_{r,l,m,s} = \arcsin\left\{\frac{a_{r,l,m,s,z}}{|\mathbf{a}_{r,l,m,s}|}\right\} \tag{71}$$

In the single-bounce model, the departure angles ϕ^d and θ^d depend on the TX and the LBS position. The initial departure angles are omitted and no longer used. Hence, when using only the single-bounce model, the departure ASs obtained from the channel coefficients will not match with the vales from the LSF model.

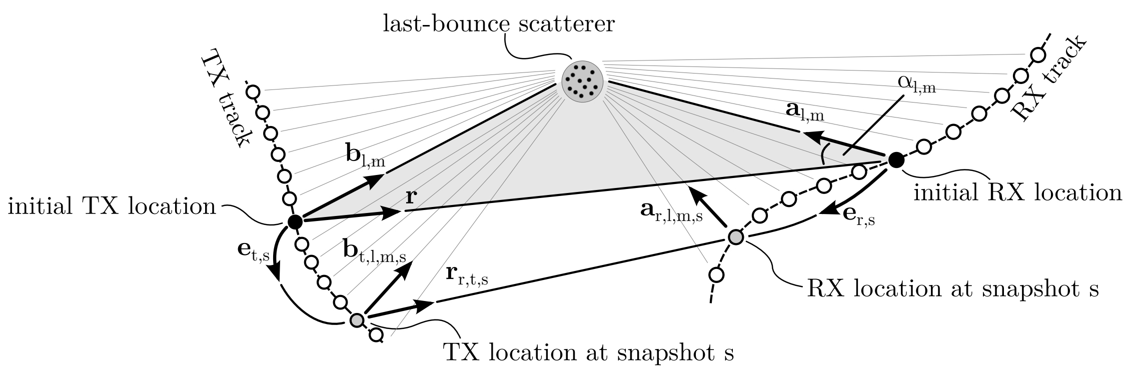


Figure 9: Illustration of the calculation of the scatterer positions and updates of the arrival angles in the single-bounce model

The angles are calculated by

$$\mathbf{b}_{t,l,m,s} = \mathbf{r} + \mathbf{a}_{l,m} - \mathbf{e}_{t,s} \tag{72}$$

$$\phi^d_{t,l,m,s} = \arctan_2 \{b_{t,l,m,s,y}, b_{t,l,m,s,x}\} \tag{73}$$

$$\theta^d_{t,l,m,s} = \arcsin \left\{ \frac{b_{t,l,m,s,z}}{|\mathbf{b}_{t,l,m,s}|} \right\} \tag{74}$$

The phases ψ and path delays τ depend on the total path length $d_{r,t,l,m,s}$. They are calculated as

$$d_{r,t,l,m,s} = |\mathbf{b}_{t,l,m,s}| + |\mathbf{a}_{r,l,m,s}| \tag{75}$$

$$\psi_{r,t,l,m,s} = \frac{2\pi}{\lambda} \cdot (d_{r,t,l,m,s} \bmod \lambda), \tag{76}$$

$$\tau_{r,t,l,s} = \frac{1}{20 \cdot c} \cdot \sum_{m=1}^{20} d_{r,t,l,m,s}. \tag{77}$$

The phase always changes with respect to the total path length. Hence, when the MT moves away from the LBS the phase will increase and when the MT moves towards the LBS the phase will decrease. This inherently captures the Doppler shift of single paths and creates a realistic Doppler spread in the channel coefficients at the output of the model.

In the next paragraph, the single-bounce model is extended to include the FBS in order to realistically model the effects caused by large array antennas at the TX. This is important in scenarios where there is scattering close to the BS, *e.g.*, indoors or in micro-cell deployments with low BS heights.

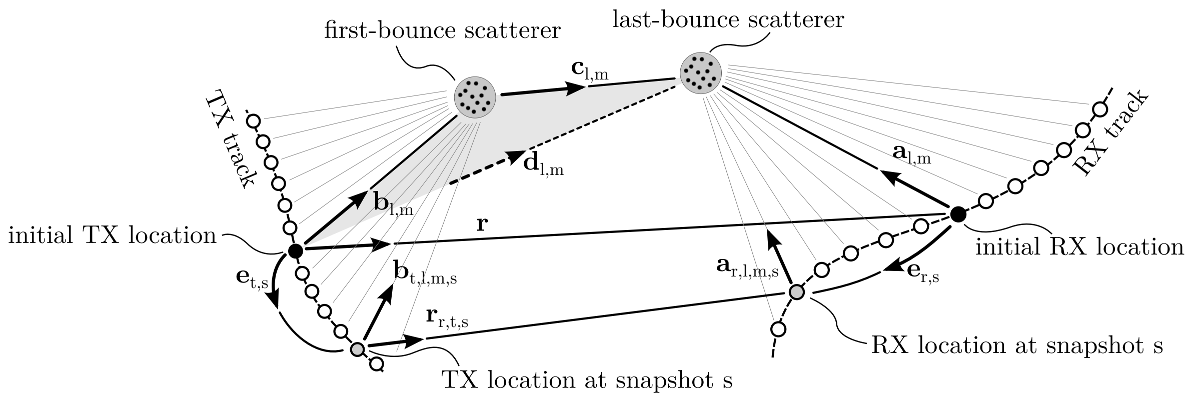

first-bounce scatterer
last-bounce scatterer
$\mathbf{c}_{\text{l,m}}$
TX track
RX track
$\mathbf{d}_{\text{l,m}}$
$\mathbf{a}_{\text{l,m}}$
$\mathbf{b}_{\text{l,m}}$
initial TX location
$\mathbf{r}$
initial RX location
$\mathbf{e}_{\text{t,s}}$
$\mathbf{b}_{\text{t,l,m,s}}$
$\mathbf{a}_{\text{r,l,m,s}}$
$\mathbf{e}_{\text{r,s}}$
$\mathbf{r}_{\text{r,t,s}}$
RX location at snapshot s
TX location at snapshot s

Figure 10: Illustration of the calculation of the scatterer positions and updates of the departure and arrival angles in the multi-bounce model

NLOS drifting (multi-bounce model) In the multi-bounce model, the NLOS propagation path consists of three parts as illustrated in Figure 10. In the first part, the vector $\mathbf{b}_{t,l,m,s}$ points from the position of a TX-antenna element t at snapshot s to the FBS. In the second part, the vector $\mathbf{c}_{l,m}$ points from the FBS to the LBS, and in the third part, the vector $\mathbf{a}_{r,l,m,s}$ points from the RX-location to the LBS. The path delay d_l given by the SSF model for the initial locations of TX and RX by (64). Hence, the total path lengths is

$$d_l = |\mathbf{b}_{l,m}| + |\mathbf{c}_{l,m}| + |\mathbf{a}_{l,m}|, \tag{78}$$

where the vector $\mathbf{b}_{l,m}$ points from the TX-array center to the FBS. The departure and arrival angles are known from the calculations in Section 3.3. Thus, the unit-length vector $\hat{\mathbf{b}}_{l,m}$ can be calculated by converting the departure angles of a path to Cartesian coordinates.

$$\hat{\mathbf{b}}_{l,m} = \begin{pmatrix} \cos\phi^d_{l,m} \cdot \cos\theta^d_{l,m} \\ \sin\phi^d_{l,m} \cdot \cos\theta^d_{l,m} \\ \sin\theta^d_{l,m} \end{pmatrix} = \frac{\mathbf{b}_{l,m}}{|\mathbf{b}_{l,m}|} \tag{79}$$

In order to explicitly calculate the positions of the FBS and the LBS, the lengths of the vectors $\mathbf{a}_{l,m}$ and $\mathbf{b}_{l,m}$ are needed. However, (78) is ambiguous. One way to fix the additional degree of freedom is to minimize the length of $\mathbf{c}_{l,m}$. Ideally, $|\mathbf{c}_{l,m}|$ becomes zero and the multi-bounce model collapses to a single-bounce model. In order to obtain realistic results, an additional minimum distance $d_{\min}$ between the phase center of the array antennas and the nearest scatterer is introduced. Then, the lengths $|\mathbf{a}_{l,m}|$ and $|\mathbf{b}_{l,m}|$ can be calculated by solving the following optimization problem

$$\begin{aligned}
\underset{|\mathbf{a}_{l,m}|,|\mathbf{b}_{l,m}|}{\text{minimize}} \quad & |\mathbf{c}_{l,m}| = d_l - |\mathbf{b}_{l,m}| - |\mathbf{a}_{l,m}| \\
\text{subject to} \quad & \mathbf{r} = \hat{\mathbf{b}}_{l,m} \cdot |\mathbf{b}_{l,m}| + \hat{\mathbf{c}}_{l,m} \cdot |\mathbf{c}_{l,m}| - \hat{\mathbf{a}}_{l,m} \cdot |\mathbf{a}_{l,m}| \,, \\
& |\mathbf{b}_{l,m}| \geq d_{\min}, \\
& |\mathbf{a}_{l,m}| \geq d_{\min}.
\end{aligned}$$

The minimum distance $d_{\min}$ might be fixed according to the scenario and center frequency, *e.g.*, 1 m for outdoor propagation and 0.1 m for indoor propagation. To solve the optimization problem $|\mathbf{a}_{l,m}|$ is set to $d_{\min}$ and $\mathbf{c}_{l,m}$ and $|\mathbf{b}_{l,m}|$ are calculated using the cosine theorem. This is possible since the TX, the FBS and the LBS form a triangle as indicated in Figure 10 by the gray shaded area.

$$\begin{aligned}
d_l^+ &= \tau_l \cdot c + |\mathbf{r}| - |\mathbf{a}_{l,m}| && (80)\\
\mathbf{d}_{l,m} &= \mathbf{r} + \hat{\mathbf{a}}_{l,m} \cdot |\mathbf{a}_{l,m}| && (81)\\
|\mathbf{b}_{l,m}| &= \frac{\left(d_l^+\right)^2 - |\mathbf{d}_{l,m}|^2}{2 \cdot \left(d_l^+ - \mathbf{d}_{l,m}^T \hat{\mathbf{b}}_{l,m}\right)} && (82)
\end{aligned}$$

There can be three possible results from this first iteration step that need to be treated separately.

1. The value obtained for $|\mathbf{b}_{l,m}|$ might be smaller than $d_{\min}$ or even smaller than 0. The latter would imply that the departure direction of the path has changed to the other side of the TX. However, this is not allowed since the departure angles are fixed. In this case, the optimization problem has no solution. This often happens when the propagation delay of a path is very short, *i.e.*, when the path is only a little longer than the LOS path.

This case is treated by setting $|\mathbf{c}_{l,m}| = 0$ and calculating new departure angles using the single-bounce model. This changes the departure angular spread. However, in most cases there will be many paths where the optimization problem has a solution. This is especially true for paths with a longer propagation delay. Hence, it is possible to mitigate the changed departure angular spread by adjusting the departure angles of the multi-bounce paths. The rationale behind this is that many measurements [6, 8] show a rather small departure angular spread of just a few degrees but a large arrival angular spread. This leaves more “room” to adjust the departure angles before they are wrapped around the unit circle. Also, paths with a short propagation delay have more power due to the exponential PDP used in (40). From a physical point of view, those paths are more likely to be scattered only once. Later paths have significantly less power and can be the result of multiple scattering events.

2. The value for $|\mathbf{b}_{l,m}|$ is larger than $d_{\min}$ and the optimization problem has a solution. In this case, there will also be an optimal solution, *i.e.*, a minimal distance $|\mathbf{c}_{l,m}|$. This minimum must be in between the start point where $|\mathbf{a}_{l,m}| = d_{\min}$ and the end point where $|\mathbf{b}_{l,m}| = d_{\min}$. For each value $|\mathbf{a}_{l,m}|$ there follows a value for $|\mathbf{b}_{l,m}|$ and a value for $|\mathbf{c}_{l,m}|$ which both can be calculated using the cosine theorem. Hence, it is possible to apply standard numeric methods such as bisection to obtain the optimal solution.

3. The optimization problem has a solution but the solution lies on one of the end points, either at $|\mathbf{a}_{l,m}| = d_{\min}$ or at $|\mathbf{b}_{l,m}| = d_{\min}$. In this case, the LBS or the FBS are very close to either the TX or the RX antenna. In this case, the problem is relaxed such that $|\mathbf{c}_{l,m}|$ is allowed to double in order to increase the space between scatterer and array antenna.

Once the positions of the FBS and the LBS are known, the departure and arrival angles are updated for each antenna element of the TX array. The arrival angles are updated using (69), (70) and (71). The depature angles are updated using (72), (73) and (74). The phases and delays are calculated using (76) and (77) where the new path length at snapshot s is

$$d_{r,t,l,m,s} = |\mathbf{b}_{t,l,m}| + |\mathbf{c}_{l,m}| + |\mathbf{a}_{r,l,m,s}| \tag{83}$$

LOS drifting The angles are calculated for each combination of TX-RX antenna elements based on the position of the element in 3-D coordinates.

$$\begin{aligned}
\mathbf{r}_{r,t,s} &= \mathbf{r} - \mathbf{e}_{t,s} + \mathbf{e}_{r,s} && (84)\\
\phi^{d}_{t,1,s} &= \arctan_2\left\{r_{r,t,s,y}, r_{r,t,s,x}\right\} && (85)\\
\theta^{d}_{t,1,s} &= \arcsin\left\{\frac{r_{r,t,s,z}}{|\mathbf{r}_{r,t,s}|}\right\} && (86)\\
\phi^{a}_{r,1,s} &= \arctan_2\left\{-r_{r,t,s,y}, -r_{r,t,s,x}\right\} && (87)\\
\theta^{a}_{r,1,s} &= \arcsin\left\{\frac{-r_{r,t,s,z}}{|\mathbf{r}_{r,t,s}|}\right\} && (88)
\end{aligned}$$

The vector $\mathbf{r}_{r,t,s}$ points from the location of the TX element t to the location of the RX element r at snapshot s. The phases and delays are determined by the length of this vector and are calculated using (76) and (77) where $d_{r,l,m,s}$ is replaced by $|\mathbf{r}_{r,t,s}|$.

At this point there is a complete description of the propagation path of each MPC, *i.e.*, the departure direction at the TX, the positions of the scatterers, the arrival direction at the RX, the phase, as well as the variation of these variables when the MT is moving. In the next Section, the antenna effects are included. This covers the orientation of the antennas at the BS and the MT, as well as the polarization effects which are closely related to the antennas.

3.5 Antennas and Polarization

One major advantage of geometry-based stochastic channel models (GSCMs) is that they allow the separations of propagation and antenna effects. Therefore, it is essential to have a description of the antenna that captures all relevant effects that are needed to accurately calculate the channel coefficients in the model. Antennas do not radiate equally in all directions. Hence, the radiated power is a function of the angle. The antenna is then defined by its directional response also known as radiation pattern. When the antenna is rotated around a fixed point, an additional variation in the amount of received power can be observed. This variation is due to the polarization of the antenna. There are various so-called polarization bases that can be used to describe this effect. Those different bases arise from the custom to define cross-polarization as “the polarization orthogonal to a reference polarization” [43]. Unfortunately, this leaves the reference polarization undefined and thus is ambiguous.

Of all the different ways to describe polarization (see [43, 56]), the polar spherical polarization basis is the most practical for GSCMs. In the polar spherical basis, the antenna coordinate system has two angles and two poles. The elevation angle θ is measured relative to the pole axis. A complete circle will go through each of the two poles, similar to the longitude coordinate in the WGS. The azimuth angle ϕ moves around the pole, similar to the latitude in WGS. Thus, the antenna is defined in *geographic* coordinates, the same coordinate system that is used in the channel model. Hence, deriving the antenna response from the previously calculated departure and arrival angles is straightforward. The electric field is resolved onto three vectors which are aligned to each of the three spherical unit vectors $\hat{\mathbf{e}}_\theta$, $\hat{\mathbf{e}}_\phi$ and $\hat{\mathbf{e}}_r$ of the coordinate system. In this representation, $\hat{\mathbf{e}}_r$ is aligned with the propagation direction of a path. In the far-field of an antenna, there is no field in this direction. Thus, the radiation pattern consists of two components, one is aligned with $\hat{\mathbf{e}}_\theta$ and another is aligned with $\hat{\mathbf{e}}_\phi$. It is usually described by a 2-element vector

$$\mathbf{F}(\theta,\phi) = \begin{pmatrix} F^{[\theta]}(\theta,\phi) \\ F^{[\phi]}(\theta,\phi) \end{pmatrix}. \tag{89}$$

The complex-valued amplitude g of a path between a transmit antenna and a receive antenna is

$$g = \sqrt{P} \cdot \mathbf{F}_r(\phi^a,\theta^a)^T \cdot \mathbf{M} \cdot \mathbf{F}_t(\phi^d,\theta^d) \cdot e^{-j\frac{2\pi}{\lambda}\cdot d}, \tag{90}$$

where $\mathbf{F}_r$ and $\mathbf{F}_t$ describe the polarimetric antenna response at the receiver and the transmitter, respectively. P is the power of the path, λ is the wavelength, d is the length of the path, (ϕ^a,θ^a) are the arrival and (ϕ^d,θ^d) the departure angles that were calculated in the previous steps. $\mathbf{M}$ is the 2×2 polarization coupling matrix. This matrix describes how the polarization changes on the way from the transmitter to the receiver. Many references (e.g. [27, 33, 57–59]) use an approximation of the polarization effects based on the XPR. The XPR quantifies the separation between two polarized channels due to different polarization orientations. $\mathbf{M}$ is then often modeled by using random coefficients ($Z_{\theta\theta}$, $Z_{\theta\phi}$, $Z_{\phi\theta}$, $Z_{\phi\phi}$) as

$$\mathbf{M} = \begin{pmatrix} Z_{\theta\theta} & \sqrt{1/\text{XPR}} \cdot Z_{\theta\phi} \\ \sqrt{1/\text{XPR}} \cdot Z_{\phi\theta} & Z_{\phi\phi} \end{pmatrix}, \tag{91}$$

where $Z \sim \exp\{j \cdot \mathcal{U}(-\pi,\pi)\}$ introduces a random phase. However, this does not account for all effects contributing to the polarization state of a radio link. For example, this model does not cover elliptical or circular polarization which depends on the phase difference between the two polarimetric components. With the above model, the phase difference is always random. Hence, the state-of-the-art GSCMs are not well suited for scenarios that rely on circular polarization such as land-mobile satellite scenarios.

A new approach on how to treat polarization in GSCMs is presented in this section. It is shown that the existing framework, *i.e.*, using radiation patterns in a polar spherical basis together with a 2×2 polarization coupling matrix, has great similarities with the Jones calculus, a method for handling polarized electromagnetic waves in the field of optics [36]. In the Jones calculus, the changes of the polarization of a electromagnetic wave are described by successive linear transformations. The same approach is used for the new channel model.

3.5.1 Relation between the Polarization Model and the Jones Calculus

R. C. Jones invented a simple method to calculate polarization effects in optics [36]. In his work, he described the polarization state of a ray of light by the so-called Jones vector. Any object that changes the polarization of the light is represented by a 2×2 Jones matrix. It was found that the product of the Jones matrix of the optical element and the Jones vector of the incident light accurately describes the polarization state of the resulting ray. Generally, this calculus can be used for any electromagnetic wave. It is especially interesting for the GSCMs such as the SCM and WINNER models where the paths are handled similarly like optical rays.

In the Jones calculus, the Jones vector contains the x and y-polarized components of the amplitude and phase of the electric field traveling along the z-direction.

$$\begin{pmatrix} E_x(t) \\ E_y(t) \end{pmatrix} = e^{j\omega t} \cdot \underbrace{\begin{pmatrix} A_x e^{j\epsilon_x} \\ A_y e^{j\epsilon_y} \end{pmatrix}}_{\text{Jones vector}} = \begin{pmatrix} J^{[\theta]} \\ J^{[\phi]} \end{pmatrix} = \mathbf{J} \tag{92}$$

The same expression is found in the antenna pattern (89) where the complex value $A_y e^{j\epsilon_y}$ from the Jones vector can be identified with the (generally also complex-valued) component $F^{[\theta]}(\theta,\phi)$ of the antenna pattern.

Likewise, $A_x e^{j\epsilon_x}$ can be identified with $F^{[\phi]}(\theta, \phi)$. This implies that the polarization coupling matrix $\mathbf{M}$ in (140) is a Jones matrix and that the Jones calculus could be easily integrated into the new channel model.

In general, $\mathbf{M}$ can be seen as a transformation operation that maps the incoming signal on the polarization plane to an outgoing signal. If the coefficients are real-valued, then linear transformations, such as rotation, scaling, shearing, reflection, and orthogonal projection as well as combinations of those operations, are possible. If the coefficients are complex-valued, then the matrix shows characteristics of a Möbius transformation. Such transformations can map straight lines to straight lines or circles and *vice versa*. Since the Jones calculus allows the use of complex-valued coefficients, it can transform linear polarized signals into circular or elliptical polarized signals and elliptical polarized signals into linear polarized signals. This implies that using (91) with complex-valued coefficients results in a completely random polarization behavior when the XPR is small, *i.e.*, when the off-diagonal elements are large. When XPR is large (and the off-diagonal elements are close to zero), then (91) describes a scaling operation.

In the next section, $\mathbf{M}$ will be calculated explicitly for the LOS and NLOS components also taking the orientation of the antennas into account. For the NLOS components, additional operations are used to convert the XPR value from the parameter tables into Jones matrices for the linear and elliptical polarization component.

3.5.2 Changing the Orientation of Antennas

The antennas are defined in their local coordinate system which is fixed when the radiation patterns are generated either by measurements or by designing the antennas using special software tools. In the channel model, orientation changes of the antennas are desirable in many cases, *e.g.*, when tilting BS arrays or changing the orientation of mobile terminals. However, such orientation changes lead to a different radiation pattern. An example is depicted in Figure 11. The left side of the figure shows an ideal dipole radiation pattern that has only an $F^{[\theta]}$ component. When the dipole gets rotated around the x-axis in Cartesian coordinates, the resulting radiation pattern will also have an $F^{[\phi]}$ component and the $F^{[\theta]}$ component is deformed. This is illustrated on the right side of the figure where the dipole is tilted by 20°. The following method shows how an existing antenna pattern can be manipulated in order to change the orientation of the antenna. Such manipulations need to take the polarization into account. It is shown that it is possible to describe this process by a 2×2 linear transformation, *i.e.*, a Jones matrix. Hence, the following method is used in the new channel model to adjust the orientation of the antennas either at the BS or at the MT by using the matrix $\mathbf{M}$ in (140). This makes the new model more flexible. For example, it is possible to use realistic radiation patterns at the MT, *e.g.*, the measured patterns from a smartphone. Then, typical orientations of the phone can be incorporated during the runtime of the channel model, *e.g.*, a user holding the phone close to the ear at a 45° angle.

When the orientation of an antenna changes, the radiation pattern has to be read at different angles (Θ, Φ) that include the effect of the orientation change. Rotations in 3-D are easier described in Cartesian coordinates. Therefore, the original angle pair (θ, ϕ) is transformed into a vector $\mathbf{c}$ that describes the arrival or departure angles in Cartesian coordinates. The three vector elements represent the x,y and z-component.

$$\mathbf{c} = \begin{pmatrix} \cos\theta \cdot \cos\phi \\ \cos\theta \cdot \sin\phi \\ \sin\theta \end{pmatrix} \tag{93}$$

A 3×3 matrix $\mathbf{R}$ can now be used to describe the orientation change in 3-D space. The example in Figure 11 was tilted by 20° around the x-axis of the coordinate system. The corresponding matrix is

$$\mathbf{R}_x(20°) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(20°) & -\sin(20°) \\ 0 & \sin(20°) & \cos(20°) \end{pmatrix}. \tag{94}$$

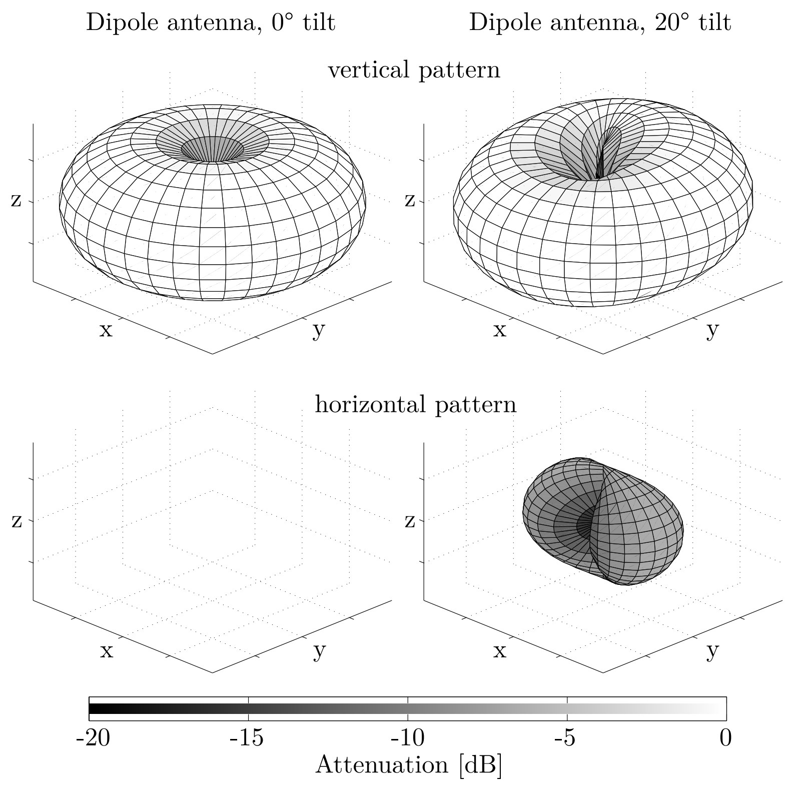


Figure 11: Example patterns for a dipole antenna

The orientation change is included in the vector $\mathbf{c}^+$ by multiplying $\mathbf{R}$ with (93).

$$\mathbf{c}^+ = \mathbf{R}^T \cdot \mathbf{c} \tag{95}$$

The transformed pattern $\tilde{\mathbf{F}}$ is needed in spherical coordinates. Thus, $\mathbf{c}^+$ is transformed back to spherical coordinates. This results in the new angles

$$\Theta = \arcsin\left[c_z^+\right], \tag{96}$$
$$\Phi = \arctan_2\left[c_y^+, c_x^+\right]. \tag{97}$$

c_x^+, c_y^+ and c_z^+ are the x, y and z component of $\mathbf{c}^+$, respectively. The coefficients of the rotated pattern are now obtained by reading the original pattern $\mathbf{F}$ at the transformed angles.

$$\hat{\mathbf{F}} = \begin{pmatrix} \hat{F}^{[\theta]} \\ \hat{F}^{[\phi]} \end{pmatrix} = \begin{pmatrix} F^{[\theta]}(\Theta,\Phi) \\ F^{[\phi]}(\Theta,\Phi) \end{pmatrix} \tag{98}$$

Since the patterns are usually sampled at a fixed angular grid, *e.g.*, at one degree resolution, interpolation is needed here since the transformed angles (Θ, Φ) will usually not be aligned with the angular grid. Linear interpolation can be used as a standard computationally inexpensive procedure.

The second step takes the polarization into account. The antenna patterns are defined in a polar-spherical polarization basis. However, the rotation is defined in Cartesian coordinates. Thus, the polarization rotation needs to be done in the Cartesian polarization basis as well. The transformation from the polar-spherical polarization basis to the Cartesian polarization basis is given by [43]

$$\begin{pmatrix} \hat{F}^{[x]} \\ \hat{F}^{[y]} \\ \hat{F}^{[z]} \end{pmatrix} = \underbrace{\begin{pmatrix} \sin\Theta\cos\Phi & -\sin\Phi \\ \sin\Theta\sin\Phi & \cos\Phi \\ -\cos\Theta & 0 \end{pmatrix}}_{=\mathbf{T}(\Theta,\Phi)} \cdot \underbrace{\begin{pmatrix} F^{[\theta]}(\Theta,\Phi) \\ F^{[\phi]}(\Theta,\Phi) \end{pmatrix}}_{=\hat{\mathbf{F}}}. \tag{99}$$

The transformation matrix $\mathbf{T}(\Theta,\Phi)$ is both orthogonal and normalized to unity. Hence, the inverse transformation matrix is equal to the matrix transpose. The rotated pattern $\tilde{\mathbf{F}}$ is obtained by using the pattern $\hat{\mathbf{F}}$ and transforming it to a Cartesian polarization basis. Then, this pattern is rotated using the rotation matrix $\mathbf{R}$ and the resulting pattern is transformed back to the polar-spherical basis. The inverse transformation needs to be done at the original angles (θ,ϕ) because the rotated antenna pattern $\tilde{\mathbf{F}}$ is aligned with the GCS used in the channel model.

$$\tilde{\mathbf{F}} = \underbrace{\mathbf{T}(\theta,\phi)^T \cdot \mathbf{R} \cdot \mathbf{T}(\Theta,\Phi)}_{=\tilde{\mathbf{M}}} \cdot \hat{\mathbf{F}} \tag{100}$$

The entire process can be described by a 2×2 polarization rotation matrix $\tilde{\mathbf{M}}$. The radiated energy in both polarization components remains constant. Hence, this matrix is a rotation matrix having the form

$$\tilde{\mathbf{M}}(\vartheta) = \begin{pmatrix} \cos\vartheta & -\sin\vartheta \\ \sin\vartheta & \cos\vartheta \end{pmatrix}, \tag{101}$$

where the polarization rotation angle ϑ follows from

$$\cos\vartheta = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ -\cos\theta \end{pmatrix}^T \cdot \mathbf{R} \cdot \begin{pmatrix} \sin\Theta\cos\Phi \\ \sin\Theta\sin\Phi \\ -\cos\Theta \end{pmatrix}, \tag{102}$$

$$\sin\vartheta = \begin{pmatrix} -\sin\phi \\ \cos\phi \\ 0 \end{pmatrix}^T \cdot \mathbf{R} \cdot \begin{pmatrix} \sin\Theta\cos\Phi \\ \sin\Theta\sin\Phi \\ -\cos\Theta \end{pmatrix}, \tag{103}$$

$$\vartheta = \arctan_2\left[\sin\vartheta, \cos\vartheta\right]. \tag{104}$$

This method provides a straightforward way to change the orientation of the antennas by

1. reading the antenna patterns at different angles (Θ, Φ) that include the orientation change,
2. calculating the polarization rotation matrix $\tilde{\mathbf{M}}$, and
3. using both to calculate the channel coefficient g in (140).

3.5.3 Constructing the Polarization Transfer Matrix

In this section, the orientation changes for the BS and MT side are combined. For the NLOS components, additional changes of the polarization are caused by scattering. The Jones calculus allows each of these effects to be modeled independently. In the end, the combined Jones matrices are used to calculate the channel coefficients.

Polarization of direct (LOS) path In the LOS polarization model, both the transmitter and the receiver can have different orientations, *e.g.*, due to a downtilt at the BS and a given movement direction at the MT. In addition, a reflection operation is needed to transform the outgoing direction at the transmitter into an incoming direction at the receiver. Thus, a combination of three linear transformations, two rotations and one reflection, is sufficient to construct the polarization transfer matrix of the LOS path.[10]

$$\begin{aligned} \mathbf{M}_{r,t,s}^{[\text{LOS}]} &= \left[\tilde{\mathbf{M}}\left(\vartheta_{r,s}^{[\text{LOS}]}\right)\right]^T \cdot \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \tilde{\mathbf{M}}\left(\vartheta_{t,s}^{[\text{LOS}]}\right) \\ &= \begin{pmatrix} \cos\vartheta_{r,s}^{[\text{LOS}]} & -\sin\vartheta_{r,s}^{[\text{LOS}]} \\ -\sin\vartheta_{r,s}^{[\text{LOS}]} & -\cos\vartheta_{r,s}^{[\text{LOS}]} \end{pmatrix} \cdot \begin{pmatrix} \cos\vartheta_{t,s}^{[\text{LOS}]} & -\sin\vartheta_{t,s}^{[\text{LOS}]} \\ \sin\vartheta_{t,s}^{[\text{LOS}]} & \cos\vartheta_{t,s}^{[\text{LOS}]} \end{pmatrix} \end{aligned} \tag{105}$$

[10]The indices denote the RX antenna element (r) and the TX antenna element (t), the path number (l), the sub-path number (m), and the snapshot number (s).

Model for the indirect (NLOS) paths For the NLOS components, the transmitted signal undergoes some diffraction, reflection or scattering before reaching the receiver. Following the common Fresnel formula in electrodynamics, the polarization direction can be changed at the boundary surface between two dielectric media. T. Svantesson [60] provided a method for modeling the polarization of a reflected wave where the polarization coupling is a function of several geometric parameters such as the orientation of the scatterers. However, these parameters are not generally available in the SCM. In addition to that, only metallic reflections keep the polarization unchanged. Reflections at dielectric media can cause changes of the polarization being a function of the complex-valued dielectric constant of the media and of the angle of incidence. Hence, not only the polarization angle might change, but also the polarization type. In order to address this issue, studies of the polarizations effects in individual scattering clusters in several outdoor- and indoor scenarios were done [37, 61, 62]. The published results indicate that scattering preserves the polarization in many cases. However, since only the powers of the elements in the polarization coupling matrix were analyzed, no conclusions can be drawn on how elliptic the polarization of the scattered wave will be.

It is possible to use the existing values for the XPR from the parameter tables of state-of-the-art GSCMs and derive additional Jones matrices in order to include the already known effects in the new channel model. The cross polarization ratio (XPR) is calculated from measurement data. A log-normal distribution is fitted to the measurement results having the average XPR_μ and the STD XPR_σ^2. When those parameters are calculated from measured data, they are usually averaged over different propagations paths. Thus, the XPR value from the parameter tables is a LSP with a scenario-dependent distribution, *i.e.*, it depends on the positions of the MT. However, here, the values $\text{XPR}_{l,m}^{[\text{dB}]}$ for individual MPCs are needed. Those are calculated in two steps. First, a value $\text{XPR}_\mu^{[\text{LSP}]}$ is drawn from

$$\text{XPR}_\mu^{[\text{LSP}]} = \mathcal{N}\left(\text{XPR}_\mu, \text{XPR}_\sigma^2\right). \tag{106}$$

This value represents the average XPR over all MPCs at the receiver positions. Then, in a second step, the XPR for the individual MPCs is drawn using $\text{XPR}_\mu^{[\text{LSP}]}$ instead of XPR_μ. This maintains the original spread XPR_σ in the generated channel coefficients.

$$\text{XPR}_{l,m}^{[\text{dB}]} = \mathcal{N}\left(\text{XPR}_\mu^{[\text{LSP}]}, \text{XPR}_\sigma^2\right) \tag{107}$$

Following the idea that the polarization coupling matrix $\mathbf{M}$ can be described by a combination of linear transformations, the model for the NLOS polarization maps the XPR to two Jones matrices, one for the linear polarization and one for the elliptic polarization. Additional deterministic components take the antenna orientations into account.

1. **Deterministic part**
   The deterministic component is the same as for the LOS polarization, *i.e.*, the different orientations of the antennas at the transmitter and the receiver are modeled by a rotation matrix as described in Section 3.5.2. A reflection operation is used to change the direction of the path.

2. **Linear component**
   During scattering, the linear polarization of a MPC might change. For example, a transmit antenna sends a *vertically* polarized wave which only oscillates in the $\hat{\mathbf{e}}_\theta$ direction. Then, a receiver might detect a wave that oscillates in both the $\hat{\mathbf{e}}_\theta$ direction and $\hat{\mathbf{e}}_\phi$ direction because scattering changed the *polarization angle* while the phases of the $\hat{\mathbf{e}}_\theta$ and $\hat{\mathbf{e}}_\phi$ components remain unchanged. In other words, a linear polarized wave stays linear polarized. In order to model this polarization change, the XPR of a path (107) is mapped to a rotation matrix. This was also suggested by [34].

$$\mathbf{M}_{l,m}^{[\text{linear}]} = \begin{pmatrix} m_{\theta\theta} & m_{\theta\phi} \\ m_{\phi\theta} & m_{\phi\phi} \end{pmatrix} = \begin{pmatrix} \cos\gamma_{l,m} & -\sin\gamma_{l,m} \\ \sin\gamma_{l,m} & \cos\gamma_{l,m} \end{pmatrix} \tag{108}$$

Following the notations in [33], the rotation angle γ is calculated as

$$\mathrm{XPR}_{l,m} = \frac{|m_{\theta\theta}|^2}{|m_{\phi\theta}|^2} = \frac{|m_{\phi\phi}|^2}{|m_{\theta\phi}|^2} = \frac{(\cos\gamma_{l,m})^2}{(\sin\gamma_{l,m})^2} = (\cot\gamma_{l,m})^2, \tag{109}$$

$$\gamma_{l,m} = \operatorname{arccot}\left(\sqrt{\mathrm{XPR}_{l,m}}\right). \tag{110}$$

3. **Elliptical component**
   When channel measurements are done with circular polarized antennas such as in land-mobile satellite scenarios [19], there is a very clear indication that scattering alters the phase between the two polarization components. In other words, a purely left hand circular polarized (LHCP) signal can be received with a right hand circular polarized (RHCP) antenna after scattering. There might also be a transformation from linear to elliptic polarization and *vice versa*. This is not covered well by the existing GSCMs. The commonly used approach in (91) creates a random phase difference between the polarization components. As a result, all paths have a (random) elliptic polarization and there is no way to adjust the XPR for circular polarized antennas. This is addressed in the new model by adding elliptic polarization using an additional Jones matrix. The phase difference between the $\hat{\mathbf{e}}_\theta$ and $\hat{\mathbf{e}}_\phi$ component is obtained by a scaling matrix

$$\mathbf{M}_{l,m}^{[\text{elliptic}]} = \begin{pmatrix} \exp(j\kappa_{l,m}) & 0 \\ 0 & \exp(-j\kappa_{l,m}) \end{pmatrix}. \tag{111}$$

   The phase shift κ is calculated using the XPR from (106) and an additional random component for each cluster.

$$\mathrm{XPR}_l^{[\text{dB}]} = \mathcal{N}\left(\mathrm{XPR}_\mu^{[\text{LSP}]}, \mathrm{XPR}_\sigma^2\right) \tag{112}$$

$$\kappa_l = \operatorname{arccot}\left(\sqrt{\mathrm{XPR}_l}\right) \tag{113}$$

   Hence, the per-cluster circular XPR is different from the per-cluster linear XPR, but all subpaths get an identical circular XPR, whereas the linear XPR differs from subpath to subpath. In this way, the same XPR can be calculated from the channel coefficients at the output of the model when using circular polarized antennas.

The polarization for the NLOS paths is a combination of five linear transformations. First, any change in the transmitter orientation is included by a rotation matrix $\tilde{\mathbf{M}}(\vartheta_{t,l,m,s})$. Then, the influence of the scattering cluster is modeled by a combination of three operations: a scaling operation that introduces a phase shift between the vertical and horizontal component to obtain an elliptic XPR, a reflection operation, and a rotation operation to obtain the desired linear XPR. Last, the change in the receiver orientation is included by a second rotation matrix $\tilde{\mathbf{M}}(\vartheta_{r,l,m,s})$. The complete polarization transfer matrix is

$$\mathbf{M}_{r,t,l,m,s}^{[\text{NLOS}]} = \left[\tilde{\mathbf{M}}(\vartheta_{r,l,m,s})\right]^T \cdot \mathbf{M}_{l,m}^{[\text{linear}]} \cdot \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \mathbf{M}_{l,m}^{[\text{elliptic}]} \cdot \tilde{\mathbf{M}}(\vartheta_{t,l,m,s}) \tag{114}$$

The equation can be simplified by combining the first three operations into one.

$$\gamma_{l,m}^{+} = \vartheta_{r,l,m,s} - \gamma_{l,m} \tag{115}$$

$$\mathbf{M}_{r,t,l,m,s}^{[\text{NLOS}]} = \begin{pmatrix} \cos\gamma_{l,m}^{+} & -\sin\gamma_{l,m}^{+} \\ -\sin\gamma_{l,m}^{+} & -\cos\gamma_{l,m}^{+} \end{pmatrix} \begin{pmatrix} \exp(j\kappa_{l,m}) & 0 \\ 0 & \exp(-j\kappa_{l,m}) \end{pmatrix} \begin{pmatrix} \cos\vartheta_{t,s} & -\sin\vartheta_{t,s} \\ \sin\vartheta_{t,s} & \cos\vartheta_{t,s} \end{pmatrix}$$

In the channel model, the polarization effects and the antenna patterns are combined into a single channel coefficient

$$g_{r,t,l,m,s}^{[1]} = \mathbf{F}_r(\Theta_{r,l,m,s}^a, \Phi_{r,l,m,s}^a)^T \cdot \mathbf{M}_{r,t,l,m,s} \cdot \mathbf{F}_t(\Theta_{t,l,m,s}^d, \Phi_{t,l,m,s}^d), \tag{116}$$

where the angle pairs (Θ^d, Φ^d) and (Θ^a, Φ^a) include the orientation of the transmit antenna element t and receive antenna element r, respectively. Contrary to (140), the phase ψ (which results from the path length) and the path power P are not included yet. They are handled separately in the next section when the sub-paths were combined into paths.

3.6 Combining Sub-Paths into Paths

Sub-paths were introduced in Section 3.3 in order to emulate fading for the NLOS MPCs over time. Each path is split into (typically 20) sub-paths. The basic assumption is that sub-paths cannot be resolved in the delay domain but have a small angular spread. Each of the sub-paths gets initialized with a random phase ψ^0. In addition, a deterministic phase component $\psi_{r,l,m,s}$ is obtained from the total length of the propagation path using (76). Both components are combined to

$$\psi^+_{r,t,l,m,s} = \exp\left(-j\psi^0_{l,m} - j\psi_{r,t,l,m,s}\right). \tag{117}$$

The initial channel coefficients for each sub-path, including the polarization and antenna effects, were calculated in the previous section. Here, the sub-paths are combined again to obtain the channel coefficients for the paths. However, due to the random initial phases, a simple sum will result in a random path power since it is impossible to predict if the phase components add up constructively or destructively. This issue is left open in WINNER and 3GPP-3D channel model. Here, it is solved by defining an average power around which the path power is allowed to fluctuate. This average value is the initial path power P_l that was calculated in Section 3.3.

In the first step, the phase (117) is combined with the initial coefficients (116) to

$$g^{[2]}_{r,t,l,m,s} = g^{[1]}_{r,t,l,m,s} \cdot \psi^+_{r,t,l,m,s}. \tag{118}$$

Then, the resulting average power is calculated for each path and each segment. Segments were introduced in Section 3.4 as part of the user trajectory along which the LSPs don't change much and where the scatterer positions remain fixed. In the above equation, the channel coefficients g are given for $s = 1 \dots S$ positions of a segment. Next, the coefficients of the 20 sub-paths are summed up and the average path powers (40) are applied.

$$g^{[3]}_{r,t,l,s} = \sum_{m=1}^{20} g^{[2]}_{r,t,l,m,s} \tag{119}$$

$$g^{[4]}_{r,t,l,s} = \sqrt{\frac{P_l}{20} \cdot \frac{\sum_{s=1}^{S}\sum_{m=1}^{20}\left|g^{[2]}_{r,t,l,m,s}\right|^2}{\sum_{s=1}^{S}\left|g^{[3]}_{r,t,l,s}\right|^2}} \cdot g^{[3]}_{r,t,l,s} \tag{120}$$

If the resulting paths are observed over time, a characteristic fluctuation of the path power can be observed, similar to measurements with limited bandwidth. If there is only one snapshot in a segment, the scaling operation (120) ensures that each path gets assigned the power value from (40). The new method ensures that the input variables given to the SSF model, *i.e.*, the delay and angular spreads, are correctly mapped to the channel coefficients generated by the model. This is different from the WINNER and 3GPP-3D channel models where the sum over the subpaths produces random path powers. Hence, the new model can also be used to create channels having specific properties, *e.g.*, a predefined DS, to benchmark algorithms. For example, it is possible to evaluate the throughput of a MIMO-orthogonal frequency division multiplexing (OFDM) scheme as a function of the DS. In the next section, the remaining LSPs, *i.e.*, the PG, the SF, and the KF are applied to the channel coefficients.

3.7 Path Gain, Shadow Fading and K-Factor

Hata [63] presented a simple model for macro-cellular settings where the PG scales with the logarithm of the distance d (in units of meters) between BS and terminal

$$\text{PG}^{[\text{dB}]} = -A \cdot \log_{10} d_{[\text{km}]} - B - C \cdot \log_{10} f_{[\text{GHz}]} + X, \tag{121}$$

where A, B and C are scenario-specific coefficients that are typically determined by measurements. The path gain exponent A often varies between values of 20 and 40, depending on the propagation conditions, the BS height, and other factors. The shadow fading (SF) is modeled by a random variable X. However, this variable is correlated with the distance between two points, *i.e.*, two closely spaced MTs will experience the same SF. A 3-D correlation model for this effect was introduced in Section 3.1 where the SF, among other parameters, is described by a map. Combining the PG and SF results in the effective path gain

$$\text{PG}_s^{[\text{eff}]} = \sqrt{10^{0.1\left(\text{PG}_s^{[\text{dB}]}+\text{SF}_s^{[\text{dB}]}\right)}}. \tag{122}$$

The movement of the MT is described by a trajectory where the index s denotes a specific position on this trajectory. Hence, the effective PG is a vector of $s = 1 \dots S$ elements. The S values of the SF in (122) come from the 3-D correlation model.

The Ricean K-factor (KF) describes the power difference between the LOS and NLOS components. In the previous section, the channel coefficients were scaled by the power values P_l that were calculated in Section 3.3.3. These power values already include the KF. However, like the SF, the KF is also spatially correlated and depends on the positions of the transmitter and receiver. The initial power values from Section 3.3.3 only consider the KF at the beginning of the trajectory. When the MT moves to a different position, its KF changes and so do the power values of the MPCs. Hence, an additional scaling factor for the path powers is needed.

$$\text{KF}_{l,s}^{[\text{scale}]} = \sqrt{1 + P_1\left(\frac{K_s}{K_0} - 1\right)} \cdot \begin{cases} \sqrt{\frac{K_s}{K_0}} & \text{for } l = 1; \\ 1 & \text{otherwise.} \end{cases} \tag{123}$$

The index $l = 1 \dots L$ is the path number, P_1 is the power of the first path that was calculated by (44), K_0 is the KF at the beginning of the trajectory, and K_s is the KF at the s^{th} position of the user trajectory. The values for K_s come from the KF map from Section 3.1. The channel coefficients from the previous section (120) are then scaled to

$$g_{r,t,l,s} = \text{PG}_s^{[\text{eff}]} \cdot \text{KF}_{l,s}^{[\text{scale}]} \cdot g_{r,t,l,s}^{[4]} \tag{124}$$

This is the last step in the small-scale-fading (SSF) model. At this point, the complex-valued amplitude g for each of the L MPCs of the CIR is described for all antenna pairs r, t at S positions of the user trajectory. In addition, there is an equal amount of values for the path delays τ that were calculated in Section 3.4. In the next section, adjacent parts of the user trajectory (*i.e.*, the segments) get merged into an even longer sequence of channel coefficients. With this, channels can be observed over long periods of time which includes transitions between propagation scenarios, *e.g.*, when a MT moves from outdoors to indoors.

3.8 Transitions between Segments

The small-scale-fading (SSF) model, which is laid out in the previous Sections 3.3 is only defined for a short part of a MT trajectory. If the MT traverses larger distances, the LSPs will change when the terminal sees different scattering clusters. Hence, in order to include long terminal trajectories in the model, there needs to be a model for the “birth and death of scattering clusters”. One idea on how to include such a process in GSCMs comes from the WINNER II model [4] where paths fade in and out over time. However, [4] does not provide a method to keep the LSPs consistent. For example, if one cluster disappears and a new on appears in its place, the delay and angular spread of the channel changes. However, those values are fixed by LSF model.

For the single-mobility case (i.e., only the RX is allowed to move, but the TX is fixed), long terminal trajectories are split into shorter segments where the LSPs are reasonably constant. Then, for each segment the small-scale-fading (SSF) model creates independent scattering clusters, channel coefficients, and path delays. Two adjacent segments are overlapping as depicted in Figure 12. The lifetime of scattering clusters

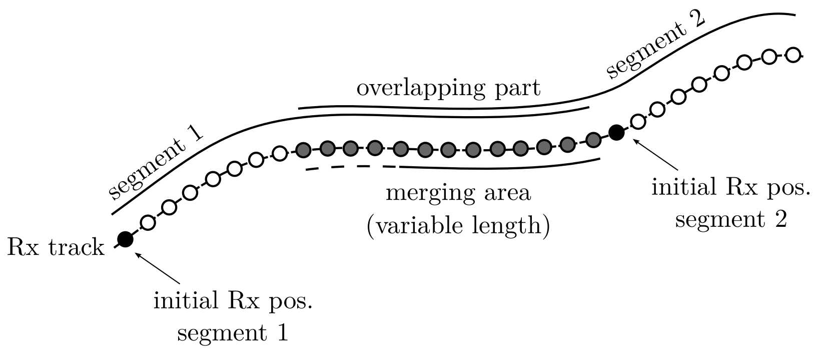


Figure 12: Illustration of the overlapping area used for calculating the transitions between segments

is confined within the combined length of two adjacent segments. In the overlapping part, the power of paths from the old segment is ramped down and the power of new paths is ramped up. Hence, this process describes the birth and death of scattering clusters along the trajectory. All paths of the segment are active outside the overlapping region. The overlapping region is further split into sub-intervals to keep the computational and memory overhead of the model low. During each sub-interval, one old path ramps down and one new path ramps up. The power ramps are modeled by a squared sine function

$$w^{[\sin]} = \sin^2\left(\frac{\pi}{2}\cdot w^{[\text{lin}]}\right). \tag{125}$$

Here, $w^{[\text{lin}]}$ is the linear ramp ranging from 0 to 1, and $w^{[\sin]}$ is the corresponding sine-shaped ramp with a constant slope at the beginning and the end. This prevents inconsistencies at the edges of the sub-intervals. If both segments have a different number of paths, the ramp is stretched over the whole overlapping area for paths without a partner. For the LOS path, which is present in both segments, only power and phase are adjusted. Paths are carefully matched to minimize the impact of the transition on the instantaneous values of the LSPs. For example, the DS increases if a path with a small delay ramps down and a similarly strong path with a large delay ramps up. Hence, the DS can fluctuate randomly within the overlapping region. To balance this out, paths from both segments are paired in a way that minimizes these fluctuations. This is done by determining the values of the DS before and after the transition. Then, a target DS is calculated for each sub-interval. For example, if the old segment yields a DS of 200 ns and the new segment has 400 ns, then the target DS will be 220 ns for the first sub-interval, 240 ns for the second and so on. Then, a combination of paths is searched that best matches the target DS for each sub-interval.

For the dual-mobility case, there are two trajectories describing both ends of the communication link, one for the RX and one for the TX. Both tracks must have the same number of snapshots and each snapshot on the RX track is paired with a snapshot on the TX track. This is illustrated in Figure 13. As for the single-mobility case, there is a birth-death process for the scattering clusters which requires the definition of segments and overlapping parts of a trajectory. However, this needs to be consistent for both sides of the communication link. For example, it is not possible that the TX is in LOS and the RX is in NLOS conditions. They either see each other, or not. For this reason, segments are only defined for the RX track. The TX track then "inherits" the segmentation. For example, when the RX track defines a new LOS segment at snapshot 22 as illustrated in Figure 13, then the TX would also start a new segment at snapshot 22 of its own track and it would also inherit the scenario definition, i.e., the new TX segment would also be a LOS segment.

3.9 Ground Reflection

As the name implies, the ground reflection (GR) is a deterministic MPC that can be received by a MT which is in direct LOS to the BS. The electromagnetic properties of the ground and the small angle of incidence usually lead to a significant part of the energy being reflected. The two paths interfere with each other,

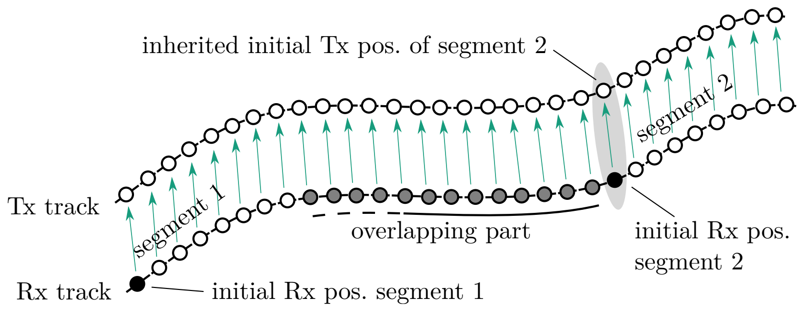

inherited initial Tx pos. of segment 2
Tx track
segment 1
segment 2
overlapping part
initial Rx pos. segment 2
Rx track
initial Rx pos. segment 1

Figure 13: Illustration of the snapshot coupling in dual-mobility simulations

causing a deterministic fading pattern. At frequencies below 6 GHz, this fading occurs only close to the BS and the distance between successive "fades" can be up to several dozen meters. Hence, GR fading was often identified as SF in past measurements and models. However, at mm-wave frequencies this effect is critical. The distance between fades can fall below 1 m and the signal strength might suddenly drop by up to 20 dB. Communication systems operating at these frequencies will also likely use narrow beams directed towards the user, but the delay and angle differences between the two components are too small to be resolved. Hence, the fades effect several GHz of bandwidth and they cannot be suppressed by beamforming. This has been confirmed by measurements at 60 GHz, where severe GR fading occurred in all measured scenarios [64, 65].

The current approach is to model this effect by a dual-slope PL model [24, 66]. At close distances between the TX and the RX, the PL is similar to the free-space loss and GR fading is approximated by additional SF. However, after a certain distance, the GR interferes destructively with the direct path which leads to an increased PL. The transition point between the two slopes, the so-called break point (BP), depends on the TX and RX heights and on the carrier frequency. The higher the carrier frequency is, the further away is the BP. At mm-wave frequencies, the BP distance is several hundred meters from the TX which means that UMi deployments with typical cell sizes below 200 m will have to cope with the fast fading effects caused by the GR.

It is possible to add a single ground reflection (GR) that dominates the multipath effect [67] to the model [13]. If the height of the RX is small compared to the distance between TX and RX, it will be difficult to resolve the GR in the delay domain. For example, in a typical UMi scenario with a TX height of 10 m, a RX height of 1.5 m and a TX-RX distance of 30 m, there are only 3.2 ns between the direct path and the reflected path. It would require more than 300 MHz of bandwidth to resolve both paths. It's power, delay, departure and arrival angles, and polarization can be explicitly calculated as described in the remainder of this section.

3.9.1 Path-Powers and Path-Delays

In order to incorporate the ground reflection, an additional path, having the delay τ_{GR} and power P_{GR}, is added to the already defined paths with delays (47) and path powers (45). It is common to use relative instead of absolute delays. Hence, the delay of the GR is calculated by

$$\tau_{\text{GR}} = \frac{\sqrt{(h_{\text{TX}} + h_{\text{RX}})^2 + d_{\text{2D}}^2} - \sqrt{(h_{\text{TX}} - h_{\text{RX}})^2 + d_{\text{2D}}^2}}{c}, \tag{126}$$

where c is the speed of light. The power of the reflected path depends on the reflection coefficient R_f which varies depending on the carrier frequency, the polarization of the incident wave and the electromagnetic properties of the ground. Hence, the reflected power is a function of the carrier frequency.

$$P_{\text{GR},f} = \frac{R_f^2}{2} \cdot P_{1,f} \tag{127}$$

A detailed discussion of the reflection coefficient is given in Section 3.9.4. In order to obtain the correct angle and delay spreads, the power of the LOS component must be adjusted to keep the normalization (45) of the path powers.

$$P_{\text{LOS},f} = \left(1 - \frac{R_f^2}{2}\right) \cdot P_{1,f} \tag{128}$$

Since the reflected path has a later delay compared to the LOS path, the DS is altered. This can be corrected by multiplying the NLOS delays with a scaling coefficient S_τ. This coefficient is calculated from

$$\sigma_\tau^2 = P_{\text{GR},f} \cdot \tau_{\text{GR}}^2 + S_\tau^2 \cdot \sum_{l=2}^{L} P_{l,f} \cdot (\tau_l)^2 - \left(P_{\text{GR},f} \cdot \tau_{\text{GR}} + S_\tau \cdot \sum_{l=2}^{L} P_{l,f} \cdot \tau_l\right)^2 \tag{129}$$

by using the initial DS σ_τ that was also used to calculate the delays in (47). Then, the final path-delays and path-powers are

$$\tau_l = \left[\, 0 \quad \tau_{\text{GR}} \quad S_\tau \tau_2 \quad \dots \quad S_\tau \tau_L \,\right], \tag{130}$$
$$P_l = \left[\, P_{\text{LOS}} \quad P_{\text{GR}} \quad P_2 \quad \dots \quad P_L \,\right]. \tag{131}$$

In the next step, the departure and arrival angles are updated in a similar way to account for the GR path.

3.9.2 Departure and Arrival Elevation Angles

In the second step, the departure and arrival angles of the ground reflection are incorporated. Four such angles are typically defined in GSCMs: the azimuth angle of departure (AoD), the azimuth angle of arrival (AoA), the elevation angle of departure (EoD), and the elevation angle of arrival (EoA). Since the azimuth angles of the GR path are identical with the LOS path and the sum-power of the LOS and GR path does not change due to (127) and (128), only elevation angles need to be considered here.

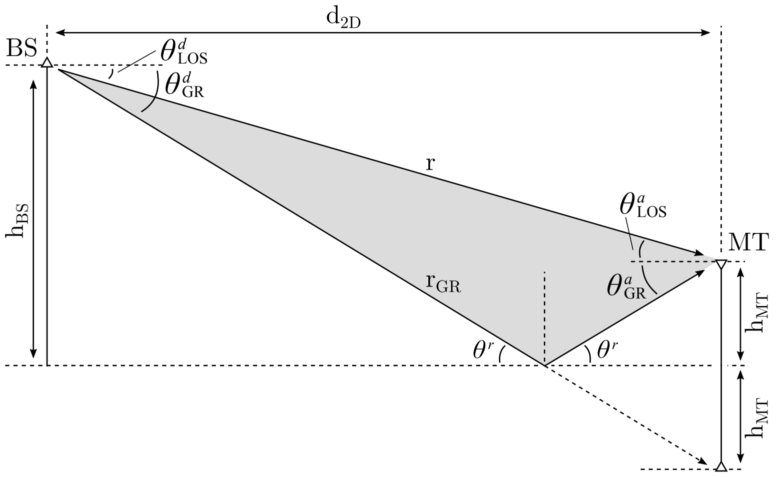


Figure 14: Illustration of the angles and vectors used for the calculations

As can be seen in Fig. 14, the elevation angle difference between the direct and the reflected path might be more significant than the delay difference. As for the path-powers and the path-delays, the angles of the NLOS paths are calculated in the channel models by (52) to achieve a predefined AS σ_θ measured in radians. The positions of the TX and RX are deterministic and so are the angles of the LOS component. The values of the angles need to be corrected to incorporate this position.

$$\theta^d_{\text{LOS}} = -\arctan\left(\frac{h_{\text{TX}} - h_{\text{RX}}}{d_{2\text{D}}}\right) \tag{132}$$

$$\theta^a_{\text{LOS}} = -\theta^d_{\text{LOS}} = \arctan\left(\frac{h_{\text{TX}} - h_{\text{RX}}}{d_{2\text{D}}}\right) \tag{133}$$

$\theta^{d/a}_{\text{LOS}}$ is the departure / arrival angle of the LOS component. The elevation angles for the GR path are deterministic as well.

$$\theta^d_{\text{GR}} = \theta^a_{\text{GR}} = -\arctan\left(\frac{h_{\text{TX}} + h_{\text{RX}}}{d_{\text{2D}}}\right) \tag{134}$$

As for the DS, the additional GR path changes the elevation angular spread, both at the TX and the RX. Hence, the angles of the other NLOS paths need to be corrected as well in order to achieve the given AS values σ^d_θ and σ^a_θ.

The AS σ_θ is defined similar to the DS where the angles get weighted by the path power [68]. However, this measure of the AS is ambiguous since the angles are distributed on a circle and the resulting value depends on the reference angle, *i.e.*, the definition of where 0° is. A linear shift $\theta_l + \Delta_\theta$ then leads to the AS being a function of Δ_θ. In the 3GPP SCM [55], this was solved by an exhaustive search over $\Delta_\theta \in [-\pi, \pi[$ to find the minimum AS

$$\sigma_\theta = \min_{\Delta_\theta} \sigma_\theta(\Delta_\theta). \tag{135}$$

Another, more efficient, approach is to normalize the angles such that the combined power-angular spectrum (PAS) of all paths points to $\theta = 0$. This normalization is done by

$$\tilde{\theta}_l = (\theta_l - \Delta_\theta + \pi \mod 2\pi) - \pi, \tag{136}$$

$$\Delta_\theta = \arg\left(\sum_{l=1}^{L} P_l \cdot \exp\left(j\theta_l\right)\right), \tag{137}$$

where $\tilde{\theta}_l$ are the normalized angles and P_l are the power values from (131). Then, the AS can be obtained by

$$\sigma_\theta = \sqrt{\sum_{l=1}^{L} P_l \cdot \left(\tilde{\theta}_l\right)^2 - \left(\sum_{l=1}^{L} P_l \cdot \tilde{\theta}_l\right)^2}. \tag{138}$$

Unfortunately, due to the normalization and the modulo operation, there is no closed form expression that can be used to calculate a scaling coefficient that corrects the angles of the NLOS paths. Hence, numerical methods must be used to determine S_θ. The updated angles then are

$$\theta_l = \left[\begin{array}{ccccc} \theta_{\text{LOS}} & \theta_{\text{GR}} & S_\theta \theta^{[2]}_2 + \theta_{\text{LOS}} & \dots & S_\theta \theta^{[2]}_L + \theta_{\text{LOS}} \end{array}\right]. \tag{139}$$

In the next step, the polarization state of the GR path is determined. This takes the dependence of the reflection coefficient on the polarization of the incident wave into account.

3.9.3 Polarization

The complex-valued amplitude g of a path between a transmit antenna and a receive antenna is

$$g = \sqrt{P} \cdot \mathbf{F}_r(\theta^a, \phi^a)^T \cdot \mathbf{M} \cdot \mathbf{F}_t(\theta^d, \phi^d) \cdot e^{-j\frac{2\pi}{\lambda}\cdot d}, \tag{140}$$

where $\mathbf{F}_r$ and $\mathbf{F}_t$ describe the polarimetric antenna responses at the receiver and the transmitter, respectively (see Section 3.5). P is the path power from (131), λ is the wavelength, d is the length of the path, (θ^a, ϕ^a) are the arrival and (θ^d, ϕ^d) the departure angles from the previous step. $\mathbf{M}$ is the 2×2 polarization coupling matrix. The LOS polarization is given by

$$\mathbf{M}_{\text{LOS}} = \left(1 - \frac{R^2}{2}\right)^{-\frac{1}{2}} \cdot \exp\left(j\psi_{\text{LOS}}\right) \cdot \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{141}$$

In this equation, the normalization (128) is removed. The 2×2 matrix can be interpreted as a reflection operation that transforms the outgoing direction of a path at the transmitter into an incoming direction at the receiver. The phase of the LOS path is

$$\psi_{\text{LOS}} = \frac{2\pi}{\lambda} \sqrt{(h_{\text{TX}} - h_{\text{RX}})^2 + d_{\text{2D}}^2}, \tag{142}$$

with λ being the carrier frequency wavelength. For the reflected path, the generally complex-valued, reflection coefficients $R_{\|}$ and $R_{\perp}$ are applied to the channel coefficients.

$$\mathbf{M}_{\text{GR}} = \frac{\sqrt{2}}{R} \cdot \exp\left(j\psi_{\text{GR}}\right) \cdot \begin{pmatrix} R_{\|} & 0 \\ 0 & -R_{\perp} \end{pmatrix}. \tag{143}$$

The factor $\frac{\sqrt{2}}{R}$ reverses the power scaling of the reflected path in (127) as well. Hence, the power of the GR path is effectively added to the CIR. However, at close BS-MT distances, the GR causes rapidly alternating constructive and destructive interference around the average signal power which leads to incorrect results if the normalization is maintained. The phase of the GR path is

$$\psi_{\text{GR}} = \frac{2\pi}{\lambda} \sqrt{(h_{\text{TX}} + h_{\text{RX}})^2 + d_{\text{2D}}^2} \tag{144}$$

Note that for dielectric materials (i.e., common ground materials), the reflection coefficients generally have negative values. Hence, there is a 180° shift between the phases the direct path and the GR path in most of the cases. In the next section, the values of the reflection coefficient are discussed.

3.9.4 Reflection Coefficient

The reflection coefficient is a function of the electromagnetic properties of a material. The complex-valued relative permittivity is given by

$$\epsilon = \epsilon_r - j\frac{\sigma}{2\pi \cdot f_c \cdot \epsilon_0} \approx \epsilon_r - j\frac{17.98 \cdot \sigma}{f_c^{[\text{GHz}]}}, \tag{145}$$

where ϵ_r is the relative permittivity and σ is the conductivity of the material. The reflection coefficients for the two polarizations are then calculated to [67, 69]

$$R_{\|} = \frac{\epsilon \cdot \sin\theta^r - Z}{\epsilon \cdot \sin\theta^r + Z} \qquad R_{\perp} = \frac{\sin\theta^r - Z}{\sin\theta^r + Z} \tag{146}$$

$$Z = \sqrt{\epsilon - \cos^2\theta^r} \tag{147}$$

$$R = \sqrt{0.5 \cdot \left|R_{\|}\right|^2 + 0.5 \cdot \left|R_{\perp}\right|^2} \tag{148}$$

$$\theta^r = -\theta_{\text{GR}}^d = \arctan\left(\frac{h_{\text{TX}} + h_{\text{RX}}}{d_{\text{2D}}}\right). \tag{149}$$

θ^r is the angle between the ground and the reflected path (see Fig. 14). An illustration of the magnitude of the reflection coefficient for a value of $\epsilon = 5$ is illustrated in Fig. 15. The average coefficient R^2 is shown as a thick black line. This value was used in (127), (128) to correct the influence of the GR path on the delay and angular spreads. The figure also shows that there is a point where only horizontally polarized waves are reflected. In optics, this corresponds to Brewster's law.

In a typical radio-propagation scenario, the values of the relative permittivity and the conductivity are frequency-dependent. A general guideline on how to model this dependency has been provided by [69], where

$$\epsilon_r = A \cdot \left(f_c^{[\text{GHz}]}\right)^B \qquad \sigma = C \cdot \left(f_c^{[\text{GHz}]}\right)^D. \tag{150}$$

[5] published curves for different ground materials. These curves have been fitted to the above model for the range from 6 to 100 GHz (see Table 36). We propose to randomly choose one of the three ground types (very dry, medium dry and wet) to determine the value of the reflection coefficient.

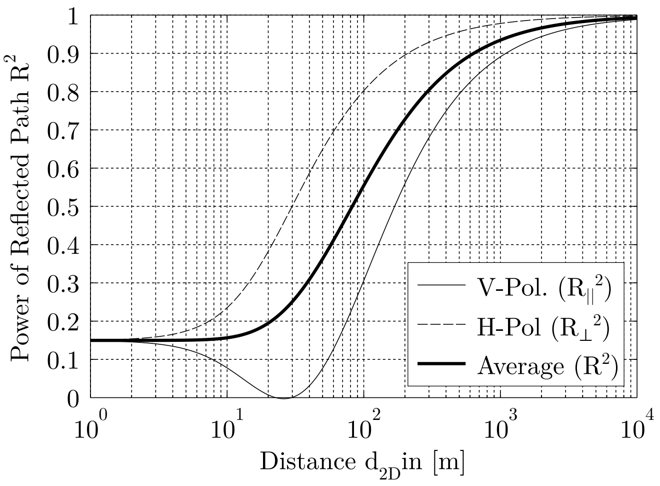


Figure 15: Values of the reflection coefficients for $\epsilon = 5$

Table 36: Electrical Properties of the Environment, 6-100 GHz [5]

	Rel. permittivity		Conductivity	
Material	A	B	C	D
Very dry ground	3	0	0.003	1.34
Medium dry ground	30.4	-0.47	0.18	1.05
Wet ground	31.3	-0.48	0.63	0.77

3.10 Summary

A new channel model has been derived from existing GSCMs such as the WINNER and 3GPP-3D model. The LSF and SSF parts of the model have been extended in several ways in order to overcome some drawbacks and limitations of the state-of-the-art approaches. The main problems that were addressed by these modifications are:

- **Spatial consistency of LSPs**
  3GPP does not specify a method to generate spatially consist random variables, neither for the LSF nor SSF model. In QuaDRiGa, a sum-of-sinusoids approach is used to correlate all random variables with the distance between two points.

- **Consistency between LSF and SSF model**
  The WINNER and 3GPP models do not map large-scale parameters to channel coefficients. They are only correct in a statistical sense. This is changed QuaDRiGa by additional scaling operations for the delays, angles, and powers after combining the sub-paths. As a result, the correct delay and angular spreads can be calculated from the generated channel coefficients of the model.

- **Mobility of MTs**
  The WINNER and 3GPP models do not allow MTs to move more than a few meters because there is no method to track the delays and directions of a path. Only the Doppler shifts of the MPCs are modeled. The mobility extensions made in QuaDRiGa are two-fold: First, a concept known as *drifting* [26] was added to the SSF model. Second, a model for the appearing and disappearing of scattering clusters was added. This is done by splitting a user trajectory in short overlapping segments. When the terminal moves from one segment to the next, the scattering clusters from the old segment are smoothly replaced with clusters from the new segment while keeping the LSPs consistent.

- **Polarization**
  The WINNER and 3GPP models do not correctly model elliptical and circular polarization. Therefore, a new model for the polarization was derived from the Jones calculus [36]. In this approach, changes

of the polarization during scattering are modeled by successive linear transformations, allowing linear and elliptic polarization to be adjusted independently.

With these updates, it is possible to generate channel coefficients with the same spatial and temporal resolution as measured data. Thus, the output of the channel model can be directly compared to the output of a measurement campaign.

4 Tutorials

This section provides a set of tutorials on how to use the channel model for different channel simulation purposes. For each of the following examples, the source code can be found in the ”tutorials” folder.

4.1 The Most Common Mistake: Handles

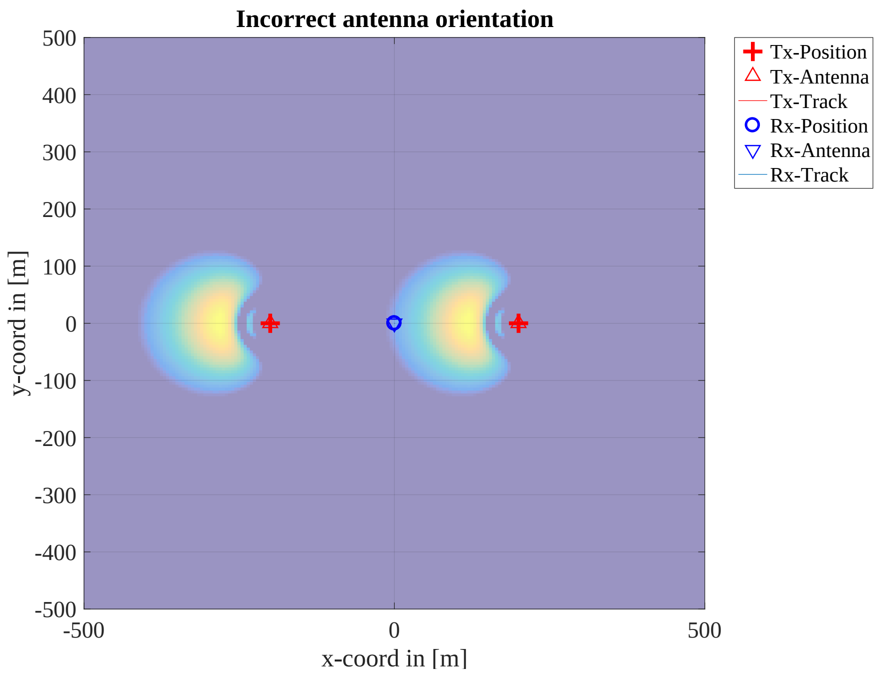
This tutorial illustrates the most common mistake that new users of the QuaDRiGa channel model often make. QuaDRiGa is implemented in MATLAB / Octave using the object-oriented framework. All QuaDRiGa objects are ”handles”. That means that a variable created from a QuaDRiGa class can be regarded as a ”pointer” to the associated data in the computer memory. This enables a very memory-efficient implementation, for example, if all mobile terminals use the same antenna. In this case, the antenna pattern only needs to be stored once in the memory and each MT only needs to store the ”pointer” to the antenna and not a copy of the data. However, working with ”handles” is something that many MATLAB users are unfamiliar with.

In the following simple example, a layout with two base stations is created. Each BS is equipped with a high-gain antenna which is tilted by 12 degrees. The antenna of the second BS is rotated by 180 degrees so that the antennas point towards each other. WARNING: The following code will not create the intended result. Try to find the mistake!

```
clear all

a = qd_arrayant('multi', 8, 0.5, 12 );                  % Generate High-Gain Antenna

l = qd_layout;                                          % New layout
l.no_tx = 2;                                            % Two BSs
l.tx_position(:,1) = [ -200 ; 0 ; 25 ];                 % Position of BS 1
l.tx_position(:,2) = [  200 ; 0 ; 25 ];                 % Position of BS 2

l.tx_array(1,1) = a;                                    % Assign antenna to BS1
l.tx_array(1,2) = a;                                    % Assign antenna to BS2
l.tx_array(1,2).rotate_pattern( 180 , 'z' );            % Rotate BS2 antenna by 180 degree
```

Here we create a plot of the layout including the sum-power that would be received by a MT at each position of the layout. You will see that the antenna of the first BS points in the wrong direction. It should point towards the east (right), but it points to the west (left).

```
close all

set(0,'defaultTextFontSize', 18)                        % Default Font Size
set(0,'defaultAxesFontSize', 18)                        % Default Font Size
set(0,'defaultAxesFontName','Times')                    % Default Font Type
set(0,'defaultTextFontName','Times')                    % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')          % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')              % Default Paper Type
set(0,'DefaultFigurePaperSize',[14.5 7.3])              % Default Paper Size

[ map,x_coords,y_coords] = l.power_map( '3GPP_38.901_UMa_LOS','quick',5,-500,500,-500,500,1.5 );
P = 10*log10(sum( abs( cat(3,map{:}) ).^2 ,3));         % Total received power

l.visualize([],[],0);                                   % Show BS and MT positions on the map
hold on
imagesc( x_coords, y_coords, P );                       % Plot the received power
hold off
axis([-500 500 -500 500])                               % Plot size
caxis( max(P(:)) + [-20 0] )                            % Color range
colmap = colormap;
colormap( colmap*0.5 + 0.5 );                           % Adjust colors to be "lighter"
set(gca,'layer','top')                                  % Show grid on top of the map
title('Incorrect antenna orientation');                 % Set plot title
```

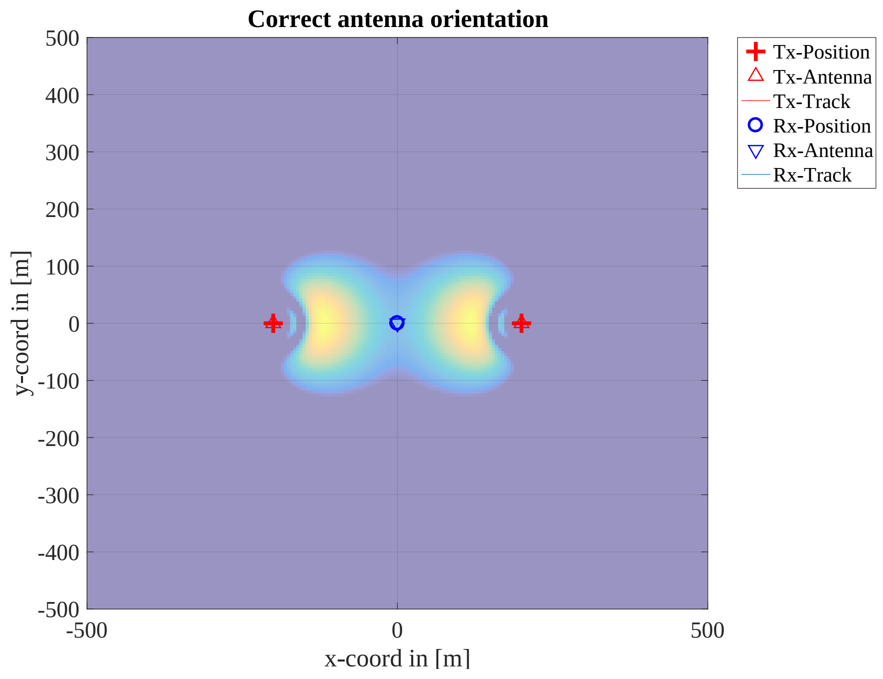


The problem is the assignment of the antenna pattern. "a", "l.tx_array(1,1)" and "l.tx_array(1,2)" point to the same object. When the rotation operation "l.tx_array(1,2).rotate_pattern" is called, the data in memory is changed. However, "a" and "l.tx_array(1,1)" point to the same object and, therefore, their properties are now changed too. The following example shows the correct way to do it. In stead of assigning a "pointer", the "copy" command creates a new object with the same data. The rotation operation only effects the antenna of BS2.

```
a = qd_arrayant('multi', 8, 0.5, 12 );                       % Generate High-Gain Antenna

l.tx_array(1,1) = copy( a );                                  % Assign copy of the antenna to BS1
l.tx_array(1,2) = copy( a );                                  % Assign copy of the antenna to BS2
l.tx_array(1,2).rotate_pattern( 180 , 'z' );                  % Rotate BS2 antenna by 180 degree
```

The following plot shows the intended result.

```
[ map,x_coords,y_coords] = l.power_map( '3GPP_38.901_UMa_LOS','quick',5,-500,500,-500,500,1.5 );
P = 10*log10(sum( abs( cat(3,map{:}) ).^2 ,3));               % Total received power

l.visualize([],[],0);                                         % Show BS and MT positions on the map
hold on
imagesc( x_coords, y_coords, P );                             % Plot the received power
hold off
axis([-500 500 -500 500])                                     % Plot size
caxis( max(P(:)) + [-20 0] )                                  % Color range
colmap = colormap;
colormap( colmap*0.5 + 0.5 );                                 % Adjust colors to be "lighter"
set(gca,'layer','top')                                        % Show grid on top of the map
title('Correct antenna orientation');                         % Set plot title
```



4.2 Typical driving course

This tutorial is a step-by-step walk through of the example given in section 1.6 of the documentation. A 800 m long drive course is covered by a S-band satellite. A car moves along the trajectory where it experiences different reception conditions. The tutorial coverers:

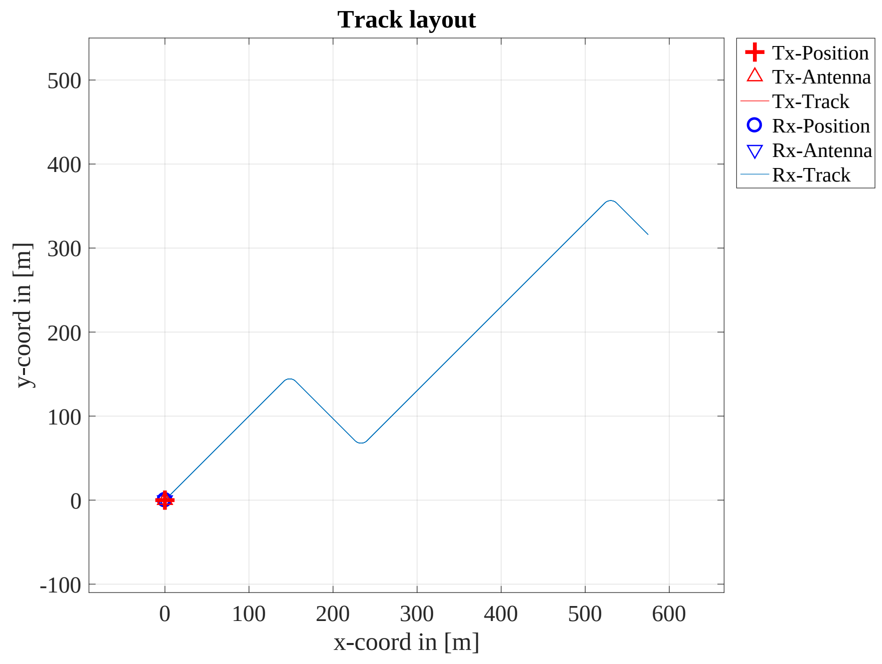
- Setting up the trajectory
- Assigning propagation environments to different sections of the track
- Modeling stops at traffic lights
- Setting up antennas for the satellite and the car
- Generating channel coefficients
- Analyzing the received power and the cross-polarization ratio

A figure showing illustrating the scenario can be found in the documentation in Section 1.6. There are 12 significant points along the track that describe an event.

1. Start environment: Urban, LOS reception of satellite signal
2. LOS to NLOS transition
3. NLOS to LOS change
4. Turning off without change in reception condition (LOS)
5. Stopping at traffic light (LOS)
6. Turning off with change of reception condition (LOS to NLOS)
7. Crossing side street (NLOS to short LOS to NLOS)
8. Structural change in the environment without a change in the environment type (higher density of buildings but still the environment remains urban)
9. Stopping at traffic lights (NLOS)
10. Houses have the same characteristics as before but are further away from the street (urban environment with different reception characteristics)
11. Change of environment (Urban to Forest)
12. Turning off without change of environment (NLOS)

Setting up the trajectory The trajectory consists of 4 straight segments of 200 m, 100 m, 400 m, and 53 m length. These segments are connected by 90 degree turns. We models these turns by arc segments having a radius of 10 m, leading to 15.7 m length. Hence, the total track length is roughly 800 meters. The following code example shows how the track can be created. In the last step, the track is plotted.

```
clear all
close all

t = qd_track('linear',200,pi/4);                            % P1-P4: 200 m segment, direction NE
t.name = 'Terminal';                                        % Set track name
t.initial_position(3,1) = 2;                                % Set the Rx height to 2 meters

c = 10*exp(1j*(135:-1:45)*pi/180);                          % P4: Turn NE to SE, 10 m curve radius
c = c(2:end)-c(1);                                          % Start relative to [x,y] = [0,0]
t.positions = [t.positions,...                              % Append curve to existing track
    [ t.positions(1,end) + real(c); t.positions(2,end) + imag(c); zeros( 1,numel(c) ) ]];

c = 100*exp( -1j*pi/4 );                                    % P4-P6: 200 m segment, direction SE
t.positions = [t.positions,...                              % Append segment to existing track
    [ t.positions(1,end) + real(c); t.positions(2,end) + imag(c); zeros( 1,numel(c) ) ]];

c = 10*exp(1j*(-135:-45)*pi/180);                           % P6: Turn SE to NE, 10 m curve radius
c = c(2:end)-c(1);                                          % Start relative to [x,y] = [0,0]
t.positions = [t.positions,...                              % Append curve to existing track
    [ t.positions(1,end) + real(c); t.positions(2,end) + imag(c); zeros( 1,numel(c) ) ]];

c = 400*exp( 1j*pi/4 );                                     % P6-P12: 400 m segment, direction NE
t.positions = [t.positions,...                              % Append segment to existing track
    [ t.positions(1,end) + real(c); t.positions(2,end) + imag(c); zeros( 1,numel(c) ) ]];

c = 10*exp(1j*(135:-1:45)*pi/180);                          % P12: Turn NE to SE, 10 m curve radius
c = c(2:end)-c(1);                                          % Start relative to [x,y] = [0,0]
t.positions = [t.positions,...                              % Append curve to existing track
    [ t.positions(1,end) + real(c); t.positions(2,end) + imag(c); zeros( 1,numel(c) ) ]];

c = 53*exp( -1j*pi/4 );                                     % P12-end: 53 m segment, direction SE
t.positions = [t.positions,...                              % Append curve to track
    [ t.positions(1,end) + real(c); t.positions(2,end) + imag(c); zeros( 1,numel(c) ) ]];

t.calc_orientation;                                         % Calculate the receiver orientation

set(0,'defaultTextFontSize', 18)                            % Default Font Size
set(0,'defaultAxesFontSize', 18)                            % Default Font Size
set(0,'defaultAxesFontName','Times')                        % Default Font Type
set(0,'defaultTextFontName','Times')                        % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')              % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')                  % Default Paper Type
set(0,'DefaultFigurePaperSize',[14.5 7.3])                  % Default Paper Size

l = qd_layout;                                              % New layout
[~,l.rx_track] = interpolate( t.copy,'distance',0.1 );      % Interpolate and assign track to layout
l.visualize([],[],0);                                       % Plot
axis equal
title('Track layout');                                      % Set plot title
```

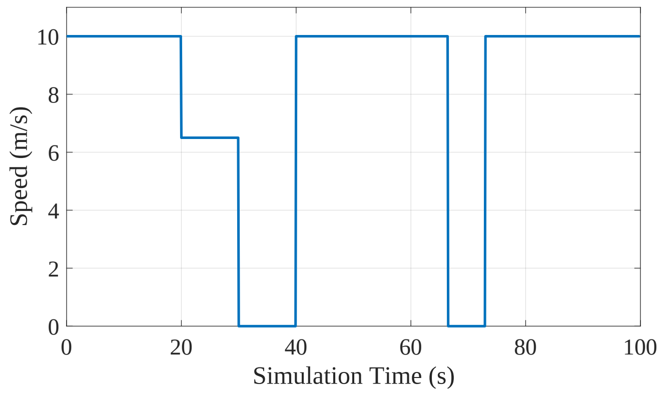


Assigning propagation environments We now assign propagation environments to the track. The easiest way to do this is by using the "add_segment" method. This method requires 3D-coordinates of a point near the track as well as a scenario description. The easiest way to obtain these coordinates is to use the data cursor in the plot and read the coordinates from the pop-up window. Scenario descriptions for satellite scenarios are provided by 3GPP TR 38.811. The propagation parameters are stored in configuration files in the QuaDRiGa source folder. Here, we only need the scenario name.

```
t.scenario{1} = '3GPP_38.881_Urban_LOS';                  % P1: Start scenario: Urban LOS
t.add_segment ([64;64;2],'3GPP_38.881_Urban_NLOS',2);     % P2: LOS to NLOS change
t.add_segment ([84;84;2],'3GPP_38.881_Urban_LOS',2);      % P3: NLOS to LOS change
t.add_segment ([233;68;2],'3GPP_38.881_Urban_NLOS',2);    % P6: LOS to NLOS change
t.add_segment ([272;103;2],'3GPP_38.881_Urban_LOS',2);    % P7: NLOS to LOS change
t.add_segment ([283;114;2],'3GPP_38.881_Urban_NLOS',2);   % P7: LOS to NLOS change
t.add_segment ([324;153;2],'3GPP_38.881_DenseUrban_NLOS',2);% P8: Higher density of buildings
t.add_segment ([420;250;2],'3GPP_38.881_Urban_NLOS',2);   % P10: Lower density of buildings
t.add_segment ([490;320;2],'3GPP_38.881_Rural_NLOS',2);   % P11: Urban to Rural
```

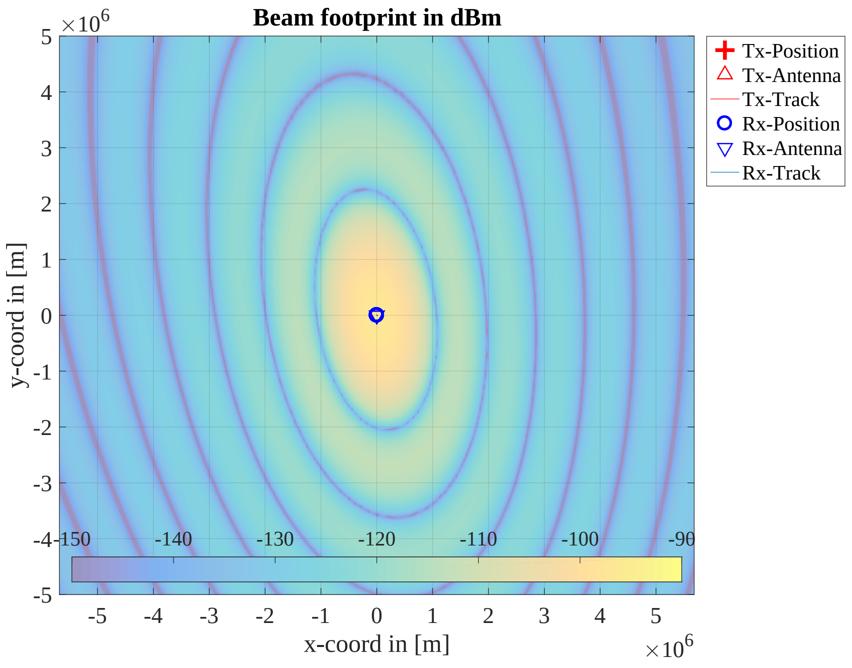
Modeling stops at traffic lights This section provides a simple way to model the movement of the car along the track. A movement profile describes the movement along the track by associating a time points (in seconds) with a traveled distance (in meters). This is assigned to the track object. The initial speed of the car is set to 10 m/s for the first 20 seconds. Then it slows down and stops after 30 seconds at the first traffic light. The stopping duration is 10 seconds. Another 6.5 second stop happens after 66.5 seconds or at 530 meters relative to the start. The total simulation time is 100 seconds. Note that accelerations are modeled. Speed changes happen suddenly as can be seen in the plot at the end of the section. For a smoother movement, it is advisable to sample the movement profile more often.

```
t.movement_profile = [ 0, 20, 30, 40, 66.5, 73, 100;... % Time points in seconds vs.
    0, 200, 265, 265, 530, 530, 800 ];                  %    distance in meters

dist  = t.interpolate('time',0.1);                      % Calculate travelled distance vs. time
time  = ( 0:numel(dist) - 2 )*0.1;                      % Calculate time sample points
speed = diff( dist ) * 10;                              % Calculate the speed

set(0,'DefaultFigurePaperSize',[14.5 4.5])              % Change Plot Size
figure('Position',[ 100 , 100 , 760 , 400]);            % New figure

plot( time,speed,'Linewidth',2 );                       % Plot speed vs. time
xlabel('Simulation Time (s)'); ylabel('Speed (m/s)'); grid on;      % Annotations
axis([0,100,0,11]);
```



Simulation layout and antenna setup This section shows how create a simulation layout, set up the center frequency and set up antennas. The system operates in S-band at a 2.2 GHz carrier frequency. The satellite uses a parabolic dish antenna of 3 m diameter, a gain of 44 dBi, and LHCP polarization. The terminal uses a dual-polarized patch antenna (LHCP/RHCP) which is pointing upwards to the sky. To verify the correct configuration, we plot the beam footprint at the end of this section. A TX power of 100 W is assumed for the satellite. This would lead to an equivalent isotropically radiated power (EIRP) of 64 dBW for the space segment. The beam footprint takes the antenna gains as well as the antenna orientation at the satellite into account. The curvature of the Earth is ignored here.

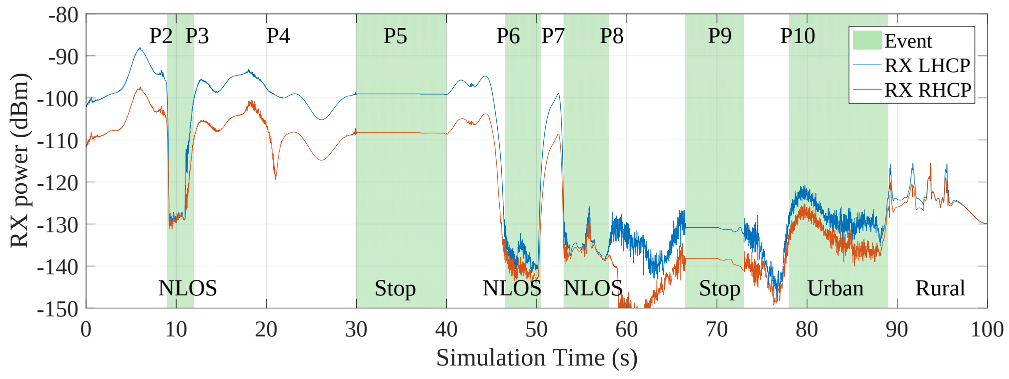
```
l = qd_layout;                                          % Create new layout
l.simpar.center_frequency = 2.2e9;                      % Set center frequency to 2.2 GHz

l.rx_track = t;                                         % Assign terminal track for the receiver
l.rx_track.split_segment(10,50,30,12);                  % Create more segments
l.rx_track.correct_overlap;                             % Fix the segment start positions

l.set_satellite_pos( 52.3, 29.7, 172.7 );               % Set GEO satellite position
l.tx_array = qd_arrayant( 'parabolic', 3, l.simpar.center_frequency, [] , 3);      % Sat. antenna
l.tx_track.orientation = [ 0 ; -29.7 ; 97.3 ]*pi/180;   % Set the orientation of tx antenna
l.tx_name{1} = 'Sat';                                   % Set TX name

l.rx_array = qd_arrayant('patch');                      % Patch antenna for the terminal
l.rx_array.center_frequency = l.simpar.center_frequency;     % Set antenna frequency
l.rx_array.copy_element(1,2);                           % Two identical elements
l.rx_array.rotate_pattern(90,'x',2);                    % Rotate second element by 90 degrees
l.rx_array.coupling = 1/sqrt(2) * [1 1 ; 1j -1j];       % Set LHCP / RHCP polarization
l.rx_array.combine_pattern;                             % Merge polarized patterns
l.rx_array.rotate_pattern(-90,'y');                     % Point skywards

% Calculate the beam footprint
set(0,'DefaultFigurePaperSize',[14.5 7.3])              % Adjust paper size for plot
[map,x_coords,y_coords]=l.power_map('3GPP_38.881_Urban_LOS','quick',2e4,-6e6,6e6,-5e6,5e6);
P = 10*log10( map{:}(:,:,1) ) + 50;                     % RX copolar power @ 50 dBm TX power
l.visualize([],[],0);                                   % Plot layout
axis([-5e6,5e6,-5e6,5e6]);                              % Axis
hold on
imagesc( x_coords, y_coords, P );                       % Plot the received power
hold off

colorbar('South')                                       % Show a colorbar
colmap = colormap;
colormap( colmap*0.5 + 0.5 );                           % Adjust colors to be "lighter"
axis equal
set(gca,'XTick',(-5:5)*1e6);
set(gca,'YTick',(-5:5)*1e6);
caxis([-150,-90])
set(gca,'layer','top')                                  % Show grid on top of the map
title('Beam footprint in dBm');                         % Set plot title
```



Generating and analyzing channel coefficients Now we generate the channel coefficients and plot the power in both polarizations over time. The plot is annotated to show the events that happen during the simulation.

```
c = l.get_channels(0.01);                                   % Generate channels

pow  = 10*log10( reshape( sum(abs(c.coeff(:,:,:,:)).^2,3) ,2,[] ) );     % Calculate the power
time = (0:c.no_snap-1)*0.01;                                % Vector with time samples

ar   = zeros(1,c.no_snap);                                  % Shading of events
ar(900:1200)  = -200;                                       % NLOS from P2 to P3
ar(3000:4000) = -200;                                       % Stop at P5
ar(4650:5050) = -200;                                       % NLOS from P6 to P7
ar(5300:5800) = -200;                                       % NLOS from P6 to P7
ar(6650:7300) = -200;                                       % Stop at P9
ar(7800:8900) = -200;                                       % Stop at P9

set(0,'DefaultFigurePaperSize',[14.5 4.5])                  % Change Plot Size
figure('Position',[ 100 , 100 , 1200 , 400]);               % New figure
a = area(time,ar,'FaceColor',[0.7 0.9 0.7],'LineStyle','none'); % Area shading
hold on; plot(time,pow'+50); hold off;
xlabel('Simulation Time (s)'); ylabel('RX power (dBm)'); grid on; axis([0,100,[-150,-80]]);
legend('Event','RX LHCP','RX RHCP'); set(gca,'layer','top')

text( 7,-85,'P2' ); text( 11,-85,'P3' ); text( 8,-145,'NLOS' );
text( 20,-85,'P4' ); text( 33,-85,'P5' ); text( 32,-145, 'Stop' );
text( 45.5,-85,'P6' ); text( 50.5,-85,'P7' ); text( 44,-145,'NLOS' );
text( 57,-85,'P8' ); text( 53,-145,'NLOS' );
text( 69,-85,'P9' ); text( 68,-145, 'Stop' );
text( 77,-85,'P10' ); text( 80,-145, 'Urban' );text( 92,-145, 'Rural' );
```

```
Starting channel generation using QuaDRiGa v2.2.0-0
1 receiver, 1 transmitter, 1 frequency (2.2 GHz)
Interpolating tracks (v = 10 m/s, SR = 100 samples/s, update factor = 1.000)
Warning: Sample density in tracks does not fulfill the sampling theoreme.
Generating channel builder objects - 4 builders, 27 channel segments
Initializing random generators
Generating parameters
SSF Corr.    [oooooooooooooooooooooooooooooooooooooooooooooooooo]     1 seconds
Channels     [oooooooooooooooooooooooooooooooooooooooooooooooooo]    60 seconds
Merging      [oooooooooooooooooooooooooooooooooooooooooooooooooo]     9 seconds
Formatting output channels - 1 channel object
Total runtime: 71 seconds
```



4.3 Effects of the Antenna-Orientation

This tutorial shows how to evaluate antenna effects. It creates a simple setup with a transmit and a receive antenna facing each other in pure LOS conditions. Then, the transmitter is rotated around its x-axis and the effect on the received power is studied.

One feature of the model is that it allows to freely orient the antennas at the transmitter and receiver. In the following, two cross-polarized patch antennas are aligned on the optical axis facing each other. The surface normal vectors of the transmit and the receive patch are aligned with the LOS. The transmitter is rotated from -90° to 90° around the optical axis. The real and imaginary parts of the channel coefficients are then calculated for each angle. Each real and imaginary part is normalized by its maximum and the results are plotted. The calculation is done for both, linearly and crossed polarized elements.

Model and Antenna Setup Here, we parametrize the simulation. We place the receiver 10 m away from the transmitter and chose the scenario "LOSonly". Thus, no NLOS components are present. The receiver is set up as a multi-element array antenna using both, circular and linear polarization.

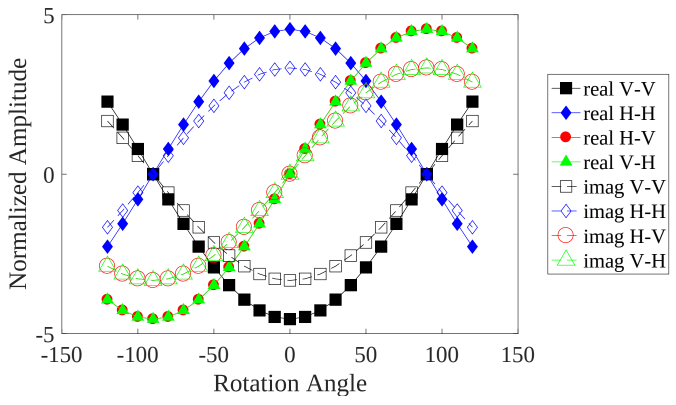
```
clear all
close all

a = qd_arrayant('lhcp-rhcp-dipole');      % Create circular polarized antenna

a2 = qd_arrayant('custom',90,90,0);       % Create linear polarized patch antenna
a2.copy_element(1,2);                     % Copy the antenna element
a2.rotate_pattern(90,'x',2);              % Rotate second element by 90 degree

a.append_array( a2 );                     % Append the second antenna to the first

l = qd_layout;
l.simpar.show_progress_bars = 0;          % Disable progress bar indicator

l.rx_track = qd_track('linear',0,pi);     % Create new track (pi turns the rx by 180 degree)
l.rx_position = [11;0;0];                 % Set the receiver position
l.tx_position = [0;0;0];

l.set_scenario( 'LOSonly' );              % Set the scenario to LOS only
l.tx_array = a;                           % Use same antenna at both sides
l.rx_array = a;
```

Iteration over all angles Next, we rotate the receive antenna in 10 degree steps around its x-axis and calculate the channel response for each angle.

```
rot = -120:10:120;                                  % Rotation angle
h = zeros(4,4,numel(rot));
for n = 1 : numel(rot)
    cc = copy( a );                                 % Create copy of the Tx antenna ( !!! )
    cc.rotate_pattern( rot(n) , 'x');                      % Assign rotation angle

    l.tx_array = cc;                                       % Set Tx antenna
    c = l.get_channels;                                    % Update channel coefficients
    h(:,:,n) = c.coeff(:,:,1,1);
end
```

Linear Polarization results Now we plot the results for the linear polarization. There are two linearly polarized antennas at the transmitter and two at the receiver. Their orientation can either be vertical (denoted as V) or horizontal (denoted as H). The channel matrix thus has 8 coefficients, VV, VH, HV and HH. Each coefficient is complex-valued. Thus, figure shows 8 curves, 4 for the real parts and 4 for the imaginary parts.

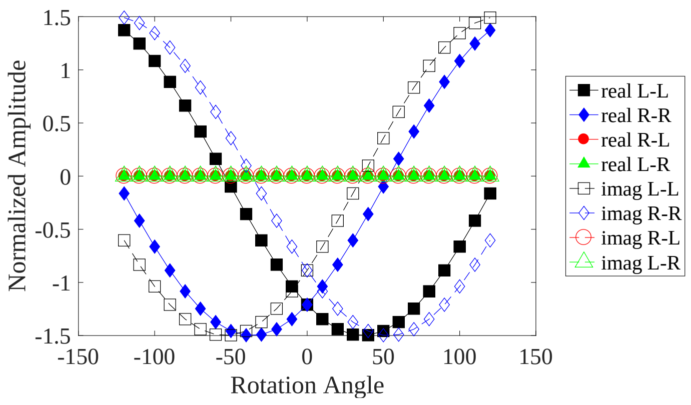
```
set(0,'defaultTextFontSize', 18)                       % Default Font Size
set(0,'defaultAxesFontSize', 18)                       % Default Font Size
set(0,'defaultAxesFontName','Times')                   % Default Font Type
set(0,'defaultTextFontName','Times')                   % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')         % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')             % Default Paper Type
set(0,'DefaultFigurePaperSize',[14.5 4.5])             % Default Paper Size

figure('Position',[ 100 , 100 , 760 , 400]);
g = h([3,4],[3,4],:);

plot(rot,squeeze(real(g(1,1,:))),'-sk','Linewidth',0.5,'MarkerfaceColor','k','Markersize',12)
hold on
plot(rot,squeeze(real(g(2,2,:))),'-db','Linewidth',0.5,'MarkerfaceColor','b','Markersize',8)
plot(rot,squeeze(real(g(2,1,:))),'-or','Linewidth',0.5,'MarkerfaceColor','r','Markersize',8)
plot(rot,squeeze(real(g(1,2,:))),'-^g','Linewidth',0.5,'MarkerfaceColor','g','Markersize',8)

plot(rot,squeeze(imag(g(1,1,:))),'--sk','Linewidth',0.5,'Markersize',12)
plot(rot,squeeze(imag(g(2,2,:))),'--db','Linewidth',0.5,'Markersize',8)
plot(rot,squeeze(imag(g(2,1,:))),'--or','Linewidth',0.5,'Markersize',12)
plot(rot,squeeze(imag(g(1,2,:))),'--^g','Linewidth',0.5,'Markersize',12)
hold off

xlabel('Rotation Angle')
ylabel('Normalized Amplitude')
legend('real V-V','real H-H','real H-V','real V-H',...
    'imag V-V','imag H-H','imag H-V','imag V-H','location','EastOutside')
```



Circular Polarization results The second plot shows the same for circular polarization. The first element is LHCP (denoted as L) and the second is RHCP (denoted as R). As expected, all cross-polarization coefficients (RL and LR) are zero.

```
figure('Position',[ 100 , 100 , 760 , 400]);
g = h([1,2],[1,2],:);

plot(rot,squeeze(real(g(1,1,:))),'-sk','Linewidth',0.5,'MarkerfaceColor','k','Markersize',12)
hold on
```

```
plot(rot,squeeze(real(g(2,2,:))),'-db','Linewidth',0.5,'MarkerfaceColor','b','Markersize',8)
plot(rot,squeeze(real(g(2,1,:))),'-or','Linewidth',0.5,'MarkerfaceColor','r','Markersize',8)
plot(rot,squeeze(real(g(1,2,:))),'-^g','Linewidth',0.5,'MarkerfaceColor','g','Markersize',8)

plot(rot,squeeze(imag(g(1,1,:))),'--sk','Linewidth',0.5,'Markersize',12)
plot(rot,squeeze(imag(g(2,2,:))),'--db','Linewidth',0.5,'Markersize',8)
plot(rot,squeeze(imag(g(2,1,:))),'--or','Linewidth',0.5,'Markersize',12)
plot(rot,squeeze(imag(g(1,2,:))),'--^g','Linewidth',0.5,'Markersize',12)
hold off

xlabel('Rotation Angle')
ylabel('Normalized Amplitude')
legend('real L-L','real R-R','real R-L','real L-R',...
    'imag L-L','imag R-R','imag R-L','imag L-R','location','EastOutside')
```



4.4 Drifting Phases and Delays

Drifting is the method used for obtaining time evolution within one segment. This tutorial demonstrates the effect of "drifting" on the channel coefficients. It shows how drifting can be enabled and disabled as well as how the resulting data can be analyzed.

Drifting is an essential feature of the channel model. Drifting enables a continuous time evolution of the path delays, the path phases, the departure- and arrival angles and the LSPs. It is thus the enabling feature for time continuous channel simulations. Although drifting was already available in the SCME branch of the WINNER channel model, it did not make it into the main branch. Thus, drifting is not available in the WIM1, WIM2 or WIM+ model. It is also not a feature of the 3GPP model family. Here the functionality is implemented again. This script focuses on the delay and the phase component of the drifting functionality.

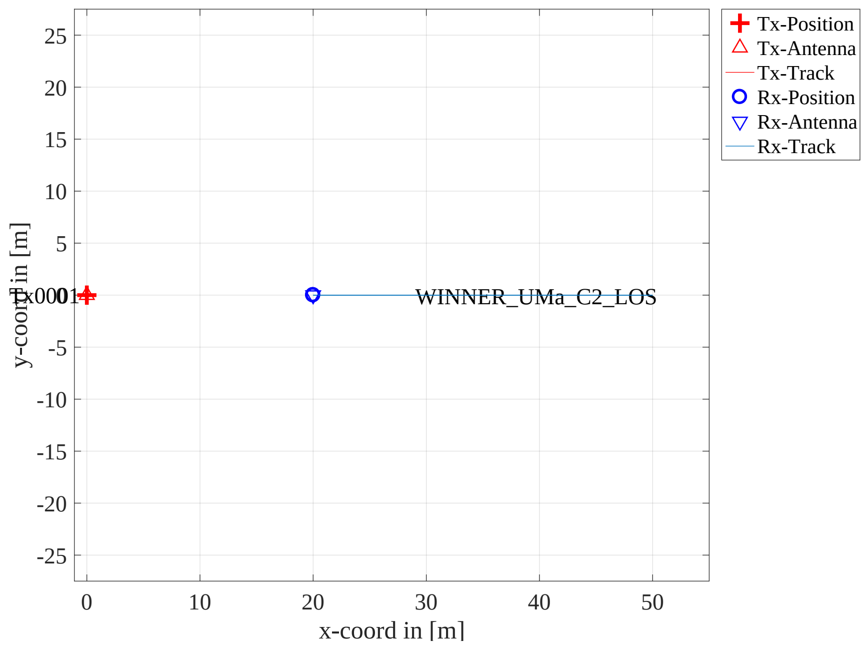
Channel model set-up and coefficient generation First, we parametrize the channel model. We start with the basic simulation parameters. For the desired output, we need two additional options: we want to evaluate absolute delays and we need to get all 20 sub-paths. Normally, the sub-paths are added already in the channel builder.

```
clear all
close all

set(0,'defaultTextFontSize', 18)                      % Default Font Size
set(0,'defaultAxesFontSize', 18)                      % Default Font Size
set(0,'defaultAxesFontName','Times')                  % Default Font Type
set(0,'defaultTextFontName','Times')                  % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')        % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')            % Default Paper Type
set(0,'DefaultFigurePaperSize',[14.5 7.3])            % Default Paper Size

s = qd_simulation_parameters;                         % New simulation parameters
s.center_frequency = 2.53e9;                          % 2.53 GHz carrier frequency
```

```
s.sample_density = 4;                                    % 4 samples per half-wavelength
s.use_absolute_delays = 1;                               % Include delay of the LOS path
s.show_progress_bars = 0;                                % Disable progress bars
```

Second, we define a user track. Here we choose a linear track with a length of 30 m. The track start 20 m east of the transmitter and runs in east direction, thus linearly increasing the distance from the receiver.

```
l = qd_layout( s );                                      % New QuaDRiGa layout
l.tx_position(3,1) = 25;                                 % 25 m BE height
l.rx_track = qd_track('linear',30,0);                    % 30 m long track facing east
l.rx_track.initial_position = [20;0;0];                  % Start position
l.set_scenario('WINNER_UMa_C2_LOS');                     % Set propagation scenario
interpolate( l.rx_track, 'distance', 1/s.samples_per_meter, [],[],1  ); % Set sampling intervals
l.visualize;                                             % Plot the layout
```



Now, we generate the LSPs. We set the shadow fading and K-factor to 1 and disable the path loss model.

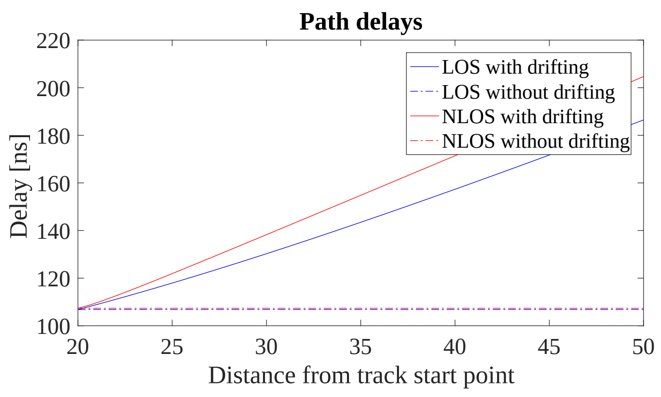
```
cb = l.init_builder;                                     % Create new builder object
cb.scenpar.SF_sigma = 0;                                 % 0 dB shadow fading
cb.scenpar.KF_mu = 0;                                    % 0 dB K-Factor
cb.scenpar.KF_sigma = 0;                                 % No KF variation
cb.plpar = [];                                           % Disable path loss model
cb.gen_parameters;                                       % Generate large- and small-scale fading
```

Now, we generate the channel coefficients. The first run uses the drifting module, the second run disables it. Note that drifting needs significantly more computing resources. In some scenarios it might thus be useful to disable the feature to get quicker simulation results.

```
cb.simpar.use_3GPP_baseline = 0;                         % Enable drifting (=spherical waves)
c = cb.get_channels;                                     % Generate channel coefficients
c.individual_delays = 0;                                 % Remove per-antenna delays

cb.simpar.use_3GPP_baseline = 1;                         % Disable drifting
d = cb.get_channels;                                     % Generate channel coefficients
```

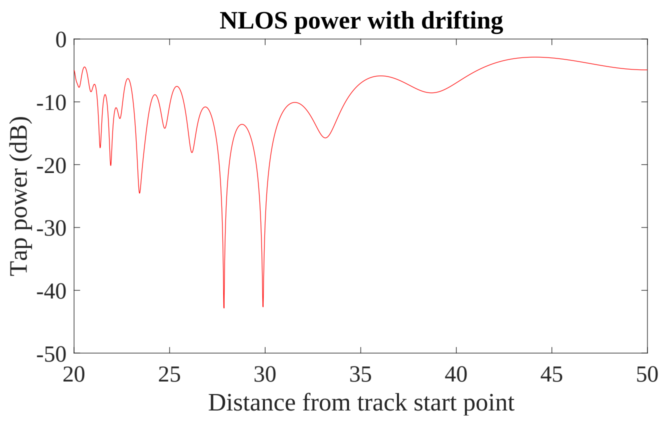
Results and discussion The following plots represent the results of the test. The first plot shows the delay of the LOS tap (blue) and the delay of the first NLOS tap (red) vs. distance. The solid lines are from the channel with drifting, the dashed lines are from the channel without. The LOS delay is always increasing since the Rx is moving away from the Tx. However, the increase is not linear due to the 25 m height of the Tx. Without drifting, the delays are not updated and stay constant during the segment. The position of the first scatterer is in close distance to the Rx (only some m away). When moving, the Rx first approaches the scatterer (delay gets a bit smaller) and then the distance increases again.

```
set(0,'DefaultFigurePaperSize',[14.5 4.5])              % Change Paper Size
figure('Position',[ 100 , 100 , 760 , 400]);            % New figure

distance = c.rx_position(1,:);                          % 2D distance between Tx and Rx
plot( distance, c.delay(1,:)*1e9 , '-b' )               % Plot LOS delay with drifting
hold on
plot( distance, d.delay(1,:)*1e9 , '-.b' )              % Plot LOS delay without drifting
plot( distance, c.delay(2,:)*1e9 , '-r' )               % Plot 1st NLOS path with drifting
plot( distance, d.delay(2,:)*1e9 , '-.r' )              % Plot 1st NLOS path without drifting
hold off
xlabel('Distance from track start point')
ylabel('Delay [ns] ')
title('Path delays')
legend('LOS with drifting','LOS without drifting','NLOS with drifting','NLOS without drifting')
```



This plot shows the power of the first NLOS tap along the track. The fading is significantly higher in the beginning and becomes much less strong towards the end.

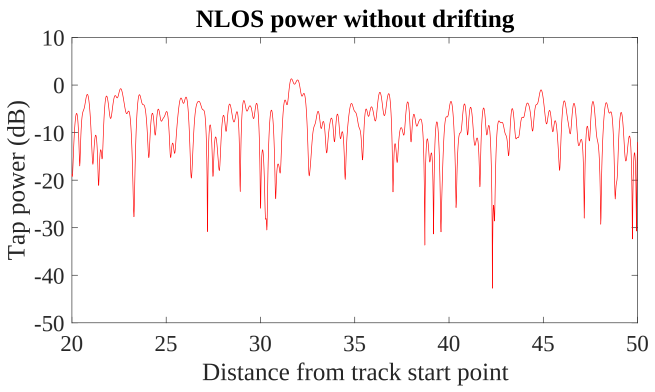
```
figure('Position',[ 100 , 100 , 760 , 400]);            % New figure
pow = abs(squeeze(sum( c.coeff(1,1,2,:,:) , 5 ))).^2;   % Calculate power of first NLOS path
plot( distance,10*log10(pow),'-r' )                     % Plot power of first NLOS path
xlabel('Distance from track start point')
ylabel('Tap power (dB)')
title('NLOS power with drifting')
```



Without drifting, the phases of the subpaths are approximated by assuming that the angles to the LBSs do not change. However, this only holds when the distance to the LBS is large. Here, the initial distance is small (ca. 5 m). When the initial angles are kept fixed along the track, the error is significant. Here, the phase ramp is negative, indicating a movement direction towards the scatterer and thus a higher Doppler frequency. However, when the scatterer is passed, the Rx moves away from the scatterer and the Doppler frequency becomes lower. This is not reflected when drifting is turned off.

Note here, that with shorter delay spreads (as e.g. in satellite channels), the scatterers are placed closer to the Rxs initial position. This will amplify this effect. Hence, for correct time evolution results, drifting needs to be turned on.

```
figure('Position',[ 100 , 100 , 760 , 400]);          % New figure
pow = abs(squeeze(sum( d.coeff(1,1,2,:,:) , 5 ))).^2;   % Calculate power of first NLOS path
plot( distance,10*log10(pow),'-r' )                     % Plot power of first NLOS path
xlabel('Distance from track start point')
ylabel('Tap power (dB)')
title('NLOS power without drifting')
```



4.5 Geometric Polarization

This tutorial shows how to study polarization effects with QuaDRiGa. Different linearly polarized antennas are defined at the transmitter and the receiver, the channel between them is calculated and the polarization effects are evaluated.

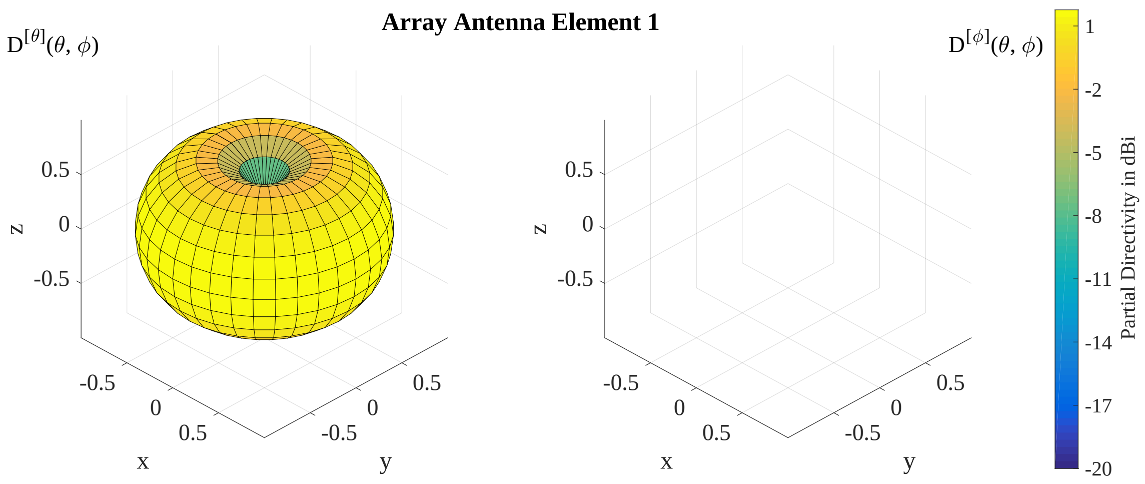
We demonstrate the polarization rotation model that calculates the path power for polarized array antennas. We do this by setting up the simulation with different H/V polarized antennas at the transmitter and at the receiver. Then we define a circular track around the receiver. When the receiver moves around the transmitter, it changes its antenna orientation according to the movement direction. In this way, all possible departure and elevation angles are sampled. Depending on the antenna orientation, the polarizations are either aligned (e.g. the Tx is V-polarized and the Rx is V-polarized), they are crossed (e.g. the Tx is V-polarized and the Rx is H-polarized), or the polarization orientation is in between those two. The generated channel coefficients should reflect this behavior.

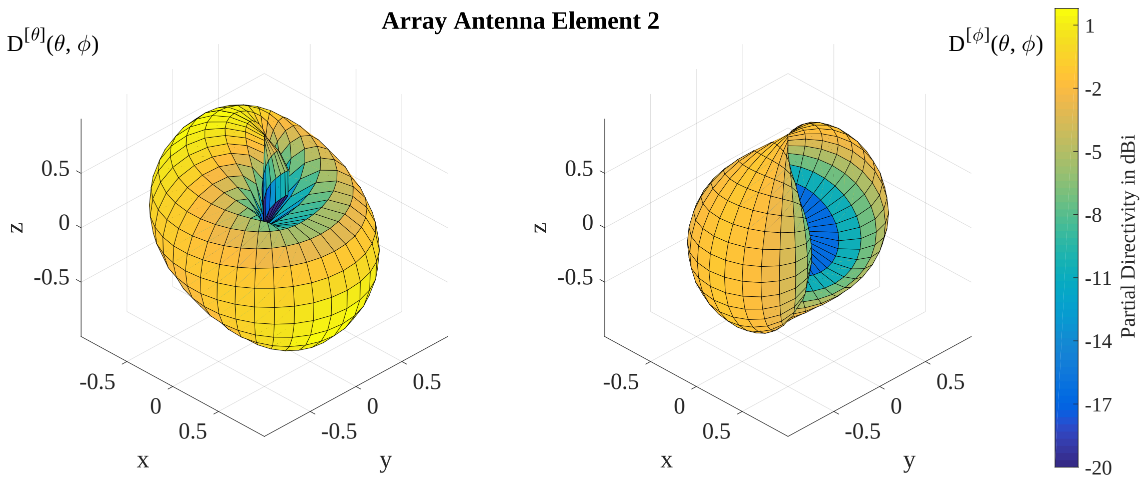
Setting up the simulation environment First, we have to set up the simulator with some default settings. Here, we choose a center frequency of 2.1 GHz. We also want to use drifting in order to get the correct angles for the LOS component and we set the number of transmitters and receivers to one.

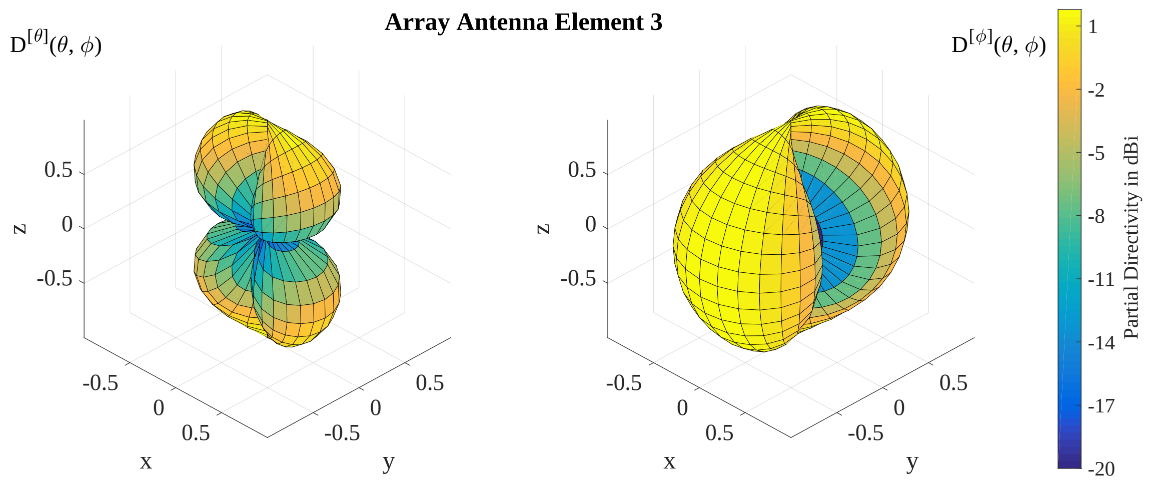
```
close all
clear all

set(0,'defaultTextFontSize', 18)                        % Default Font Size
set(0,'defaultAxesFontSize', 18)                        % Default Font Size
set(0,'defaultAxesFontName','Times')                    % Default Font Type
set(0,'defaultTextFontName','Times')                    % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')          % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')              % Default Paper Type

s = qd_simulation_parameters;                           % Set the simulation parameters
s.center_frequency = 2.1e9;                             % Center-frequency: 2.1 GHz
s.samples_per_meter = 360/(40*pi);                      % One sample per degree
s.show_progress_bars = 0;                               % Disable progress bars
```

Setting up the array antennas In the second step, we set up our array antennas. We use the synthetic dipole antennas for this case. Those antennas show perfect polarization characteristics. First, we generate a single dipole with V-polarization. Then, we create multiple copies of this antenna element and rotate them by 45 and 90 degrees, respectively. We then use the same array antenna for the receiver.

```
l = qd_layout( s );                                           % Create a new Layout
l.tx_array = qd_arrayant('dipole');                           % create V-polarized dipole
l.tx_array.set_grid( (-180:10:180)*pi/180 , (-90:10:90)*pi/180 );
l.tx_array.Fa = l.tx_array.Fa ./ max(l.tx_array.Fa(:));

l.tx_array.copy_element(1,2:3);                               % Duplicate the elements
l.tx_array.rotate_pattern(45,'y',2);                          % 45 degree polarization
l.tx_array.rotate_pattern(90,'y',3);                          % 90 degree polarization
l.rx_array = l.tx_array;                                      % Use the same array for the Rx

set(0,'DefaultFigurePaperSize',[14.5 5.3])                    % Adjust paper size for plot
l.tx_array.visualize(1);pause(1);                             % Plot the first antenna element
l.tx_array.visualize(2);pause(1);                             % Plot the second antenna element
l.tx_array.visualize(3);pause(1);                             % Plot the third antenna element
```

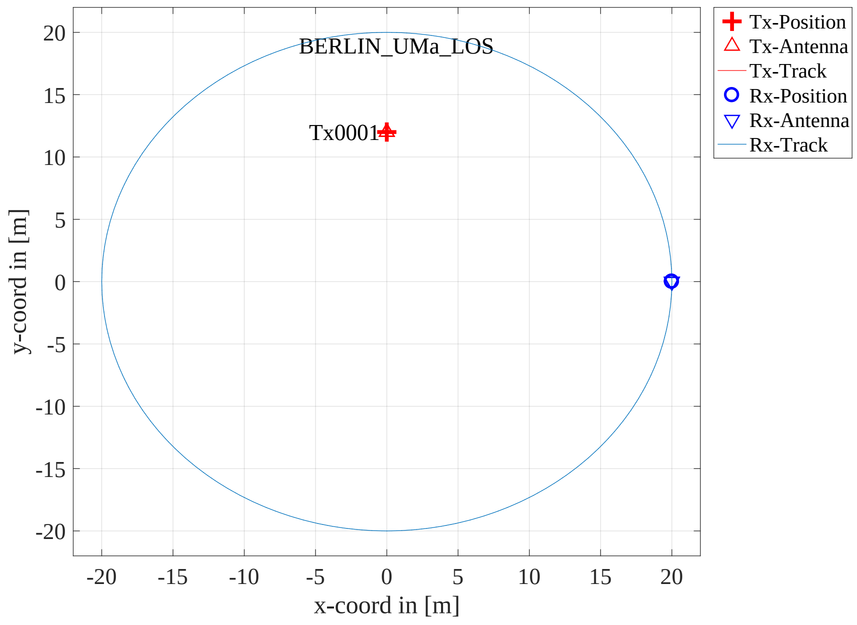






Defining a track The third step defines the track. Here, we use a circle with 40 m diameter starting in the east, traveling north. We also choose a LOS scenario since we want to study the LOS polarization. The transmitter is located 12 m north of the center of the circle at an elevation of 6 m.

```
l.rx_track = qd_track('circular',40*pi,0);                    % Circular track, radius 20 m
interpolate_positions( l.rx_track, s.samples_per_meter );       % Interpolate positions
l.tx_position = [ 0 ; 12 ; 6 ];                               % Tx position
l.rx_position = [ 20 ; 0 ; 0 ];                               % Start position for the Rx track
l.set_scenario('BERLIN_UMa_LOS');

set(0,'DefaultFigurePaperSize',[14.5 7.3])                    % Adjust paper size for plot
l.visualize;                                                  % Plot the layout
```



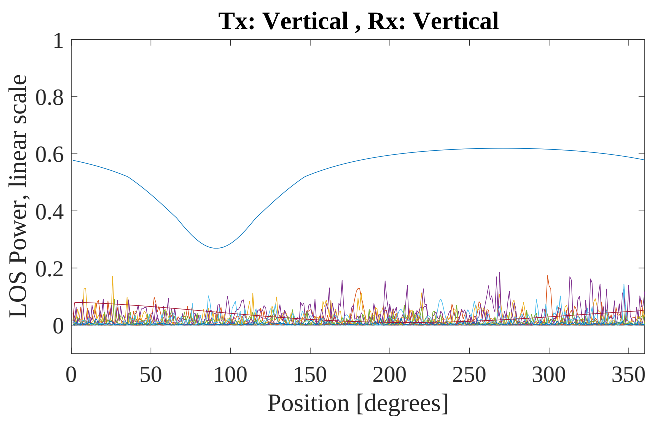
Generating channel coefficients Now, we have finished the parametrization of the simulation and we can generate the channel coefficients. We thus create a new set of correlated LSPs and the fix the shadow fading and the K-factor to some default values. This disables the drifting for those parameters. We need to do that since otherwise, drifting and polarization would interfere with each other.

```
cb = l.init_builder;                                          % Create parameter sets
cb.scenpar.KF_mu = 3;                                         % Fix KF to 3 dB
cb.scenpar.KF_sigma = 0;
cb.scenpar.SF_sigma = 0;                                    % Fix SF to 0 dB
cb.plpar = [];                                              % Disable path loss model

cb.gen_parameters;                                          % Generate small-scale-fading
c = cb.get_channels;                                        % Get the channel coefficients
```

Results and Evaluation We now check the results and confirm, if they are plausible or not. We start with the two vertically polarized dipoles at the Tx and at the Rx side. The model creates 15 taps, which is the default for the "BERLIN_UMa_LOS" scenario. Without path-loss and shadow fading (SF=1), the power is normalized such that the sum over all taps is 1 W and with a K-Factor of 3 dB, we get a received power of 0.67W for the LOS component. The remaining 0.33 W are in the NLOS components. The results can be seen in the following figure.

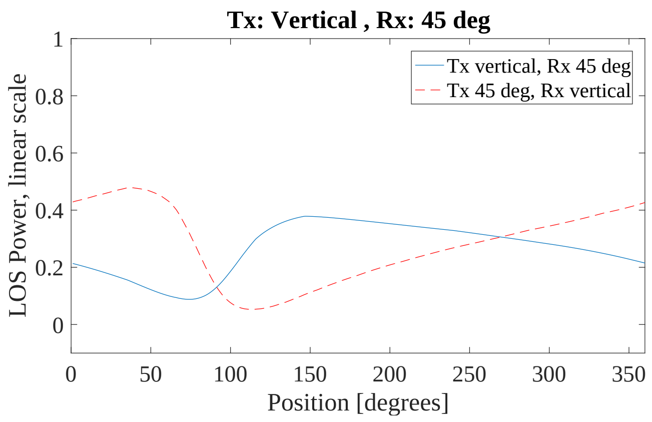
```
set(0,'DefaultFigurePaperSize',[14.5 4.5])                  % Change Paper Size
figure('Position',[ 100 , 100 , 760 , 400]);                % New figure

plot(abs(squeeze( c.coeff(1,1,:,:) )').^2);                 % Plot the graph
axis([0 360 -0.1 1]);                                       % Set the axis
xlabel('Position [degrees]');                               % Add description
ylabel('LOS Power, linear scale');
title('Tx: Vertical , Rx: Vertical');                       % Add title

disp(['LOS power:  ',num2str(mean( abs(c.coeff(1,1,1,:)).^2 , 4))])
disp(['NLOS power: ',num2str(mean( sum(abs(c.coeff(1,1,2:end,:)).^2,3) , 4))])
```

```
LOS power:  0.52832
NLOS power: 0.18903
```



The LOS power is almost constant when the Rx is south of the Tx. However, in close proximity (at 90 degree), the power is lowered significantly. This comes from the 6 m elevation of the Tx. When the Rx is almost under the Tx, the radiated power of the Dipole is much smaller compared to the broadside direction. The average power of the LOS is thus also lowered to 0.56 W. The average sum-power if the 7 NLOS components is 0.26 W. This mainly come from the XPR which leakes some power from the vertical- into the horizontal polarization and thus reduces the received power on the vertically polarized Dipole. Next, we study two cases. Either the Tx is vertical polarized and the Rx is at 45 degree or vise versa.

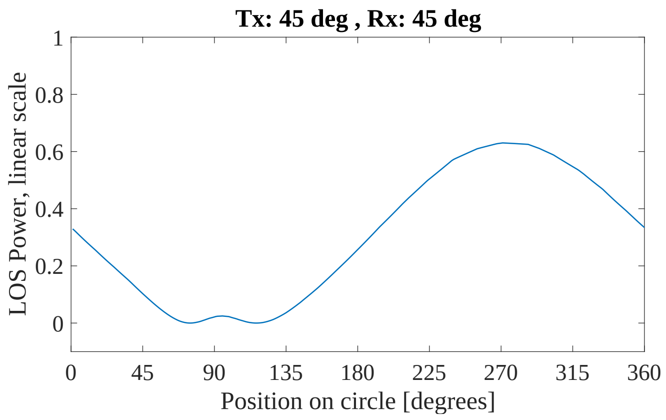
```
figure('Position',[ 100 , 100 , 760 , 400]);                % New figure
plot(abs(squeeze( c.coeff(2,1,1,:) )).^2);                  % Tx vertical, Rx 45 degree
hold on
plot(abs(squeeze( c.coeff(1,2,1,:) )).^2,'--r');            % Tx 45 degree, Rx vertical
hold off
axis([0 360 -0.1 1]);
legend('Tx vertical, Rx 45 deg', 'Tx 45 deg, Rx vertical')
xlabel('Position [degrees]');
ylabel('LOS Power, linear scale');
title('Tx: Vertical , Rx: 45 deg');
```



The receiver changes its direction in a way that it always has the same orientation towards the Tx. However, due to the displacement of the Tx, the radiated power towards the Tx becomes minimal at around 90 degree. This minimum is visible in both curves (blue and red). However, the pole of the 45 degree slanted dipole now points to a different direction which explains the difference in the two lines. When the Rx is at 45 degeee and the Tx is vertical, the pole is in the right half if the circle - resulting in a lower received power. When the Rx is Vertical and the Tx is 45 degree, the minimum power is achieved in the left half of the circle.

Next, we evaluate the two dipoles which are rotated by 45 degree. When moving around the circle, the Tx stays fixed and the Rx rotates. Subsequently, at one position, we will have both dipoles aligned and at another position, both will be crossed. When they are crossed, the received power will be 0 and when they are aligned, the power will match the first plot (two vertical dipoles). This can be seen in the following figure.

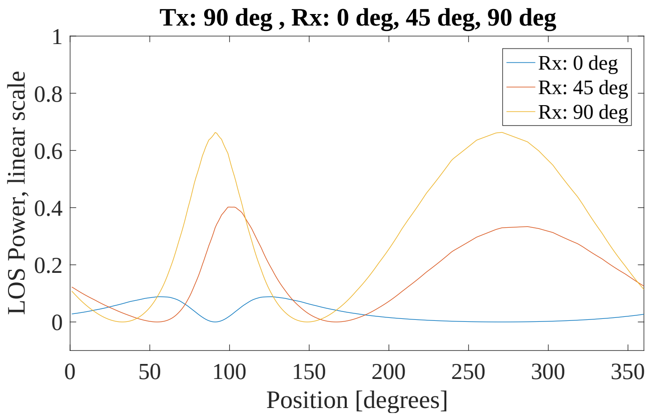
```
figure('Position',[ 100 , 100 , 760 , 400]);             % New figure
plot(abs(squeeze( c.coeff(2,2,1,:) )).^2 , 'Linewidth',1);
axis([0 360 -0.1 1]);
set(gca,'XTick',0:45:360)
xlabel('Position on circle [degrees]');
ylabel('LOS Power, linear scale');
title('Tx: 45 deg , Rx: 45 deg');
```



In the last figure, we have the Tx-antenna turned by 90 degree. It is thus lying on the side and it is horizontally polarized. For the Rx, we consider three setups: Vertical (blue line), 45 degree (green line) and 90 degree (red line). Note that the Tx is rotated around the y-axis. At the initial position (0 degree), the Rx (45 and 90 degree) is rotated around the x-axis. This is because the movement direction.

```
figure('Position',[ 100 , 100 , 760 , 400]);             % New figure
plot(abs(squeeze( c.coeff(:,3,1,:) ))'.^2);
axis([0 360 -0.1 1]);
```

```
legend('Rx: 0 deg','Rx: 45 deg','Rx: 90 deg' )
xlabel('Position [degrees]');
ylabel('LOS Power, linear scale');
title('Tx: 90 deg , Rx: 0 deg, 45 deg, 90 deg');
```



When the receiver is vertical (blue line), both antennas are always crossed. There is no position around the circle where a good link can be established. When the receiver is horizontal (red line), however, there are two points where the two dipoles are aligned. For the 45 degree dipole, the same behavior can be observed but with roughly half the power.

4.6 Pairing and segments

This tutorial shows how to set up scenarios with several transmitters and receivers and the use of scenarios. First, we set up a basic simulation with two transmitters. One of them is outdoors, the other is indoors.

```
clear all

s = qd_simulation_parameters;                          % Set up simulation parameters
s.show_progress_bars = 0;                              % Disable progress bars
s.center_frequency = 2.53e9;                           % Set center frequency
l = qd_layout(s);                                      % Create new QuaDRiGa layout

l.no_tx = 2;                                           % Two BSs
l.tx_position(:,1) = [ -142 ; 355 ; 64 ];              % Outdoor BS
l.tx_position(:,2) = [ 5 ; 0; 10 ];                    % Indoor BS
```

We create two different MTs. MT1 is indoors. The link to BS1 is in scenario ”WINNER_UMa_C2_NLOS”. The link to BS2 is in ”WINNER_Indoor_A1_LOS”. MT1 has no segments. The rows in ”qd_track.scenario” indicate the scenario for each BS. If there is only one row, then all BSs get the same scenario. The second MT is outdoors, far away from the indoor BS. The first part of the MT2 track is in LOS, the second is in NLOS. The columns of track.scenario indicate the segments. Here, all BSs get the same scenarios.

```
l.no_rx = 2;                                           % Two MTs
l.rx_track(1,1) = qd_track('linear', 0.2 );            % Linear track with 20 cm length
l.rx_track(1,1).name = 'Rx1';                          % Set the MT1 name
l.rx_track(1,1).scenario = {'WINNER_UMa_C2_NLOS';'WINNER_Indoor_A1_LOS'};  % Two Scenarios

l.rx_track(1,2) = qd_track('linear', 0.2 );            % Linear track with 20 cm length
l.rx_track(1,2).name = 'Rx2';                          % Set the MT2 name

l.rx_position(:,2) = [ 100;50;0 ];                     % Start position of the MT2 track
interpolate_positions( l.rx_track, s.samples_per_meter );  % Interpolate positions

l.rx_track(1,2).segment_index = [1 3];                 % Set segments
l.rx_track(1,2).scenario = {'WINNER_UMa_C2_LOS','WINNER_UMa_C2_NLOS'};
```

We calculate the channel coefficients and plot the list of created segments.

```
cb = l.init_builder;                                        % Initialize builder
gen_parameters( cb );                                       % Generate small-scale-fading
c = get_channels( cb );                                     % Get channel coefficients
disp( strvcat( c.name ) )                                   % Show the names if the channels
```

```
WINNER-UMa-C2-NLOS_Tx0001_Rx1
WINNER-UMa-C2-NLOS_Tx0001_Rx2_seg0002
WINNER-UMa-C2-LOS_Tx0001_Rx2_seg0001
WINNER-UMa-C2-NLOS_Tx0002_Rx2_seg0002
WINNER-UMa-C2-LOS_Tx0002_Rx2_seg0001
WINNER-Indoor-A1-LOS_Tx0002_Rx1
```

As we can see, 6 segments were generated. However, the channel Tx2_Rx2 will most likely not be needed because of the large distance. We thus remove the link from the pairing matrix and recompute the channels.

```
l.pairing = [1 2 1 ; 1 1 2 ];                               % Change the pairing matrix

cb = l.init_builder;                                        % Initialize channel builder object
gen_parameters( cb );                                       % Generate small-scale-fading parameters
c = get_channels( cb );                                     % Get channel coefficients
disp( strvcat( c.name ) )                                   % Show the names of the channels
```

```
WINNER-UMa-C2-NLOS_Tx0001_Rx1
WINNER-UMa-C2-NLOS_Tx0001_Rx2_seg0002
WINNER-UMa-C2-LOS_Tx0001_Rx2_seg0001
WINNER-Indoor-A1-LOS_Tx0002_Rx1
```

At last, we can combine the segments and generate the final channels.

```
cn = merge( c );                                            % Combine the channel coefficients
disp( strvcat( cn.name ) )                                  % Show the names if the channels
```

```
Merging      [oooooooooooooooooooooooooooooooooooooooooooooooooo]     0 seconds
Tx0001_Rx1
Tx0001_Rx2
Tx0002_Rx1
```

4.7 Network Setup and Parameter Generation

The tutorial demonstrates how to setup a simple layout with multiple receivers, how to adjust parameters manually, generate channel coefficients, and how to calculate parameters from the data. The channel model class 'qd_builder' generates correlated values for the LSPs. The channel builder then uses those values to create coefficients that have the specific properties defined in the builder objects. One important question is therefore: Can the same properties which are defined in the builder also be found in the generated coefficients? This is an important test to verify, if all components of the channel builder work correctly.

Channel model setup and coefficient generation We first set up the basic parameters.

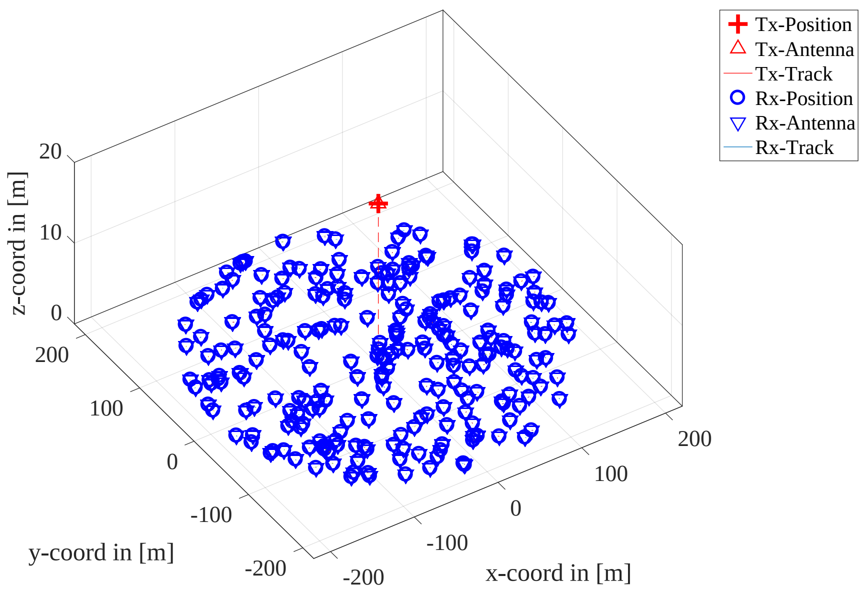
```
close all
clear all

set(0,'defaultTextFontSize', 18)                            % Default Font Size
set(0,'defaultAxesFontSize', 18)                            % Default Font Size
set(0,'defaultAxesFontName','Times')                        % Default Font Type
set(0,'defaultTextFontName','Times')                        % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')              % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')                  % Default Paper Type

s = qd_simulation_parameters;                               % Set up simulation parameters
s.show_progress_bars = 1;                                   % Show progress bars
s.center_frequency = 2.53e9;                                % Set center frequency
s.samples_per_meter = 1;                                    % 1 sample per meter
s.use_absolute_delays = 1;                                  % Include delay of the LOS path
```

Receive antenna In order to verify the angular spreads, we need to calculate the angles and the resulting angular spreads from the channel coefficients. However, the arrival angle information is embedded in the channel coefficients. In order to obtain the angles, we need a special antenna that allows us to calculate the arrival angles from the channel response. Such an "ideal" antenna is generated here. It consists of 31 elements that allow us to calculate the azimuth and elevation direction of a path as well as the polarization.

```
[ theta, phi, B, d_phi ] = qf.pack_sphere( 27 );          % Generate equidistant directions
N = numel( theta );                                        % Store number of directions
a = qd_arrayant('custom',20,20,0.05);                      % Main beam opening and front-back ratio
a.element_position(1) = 0.2;                               % Element distance from array phase-center
a.copy_element(1,2:N+3);                                   % Set number of elements
for n = 1:N                                                % Create sub-elements
    a.rotate_pattern( theta(n)*180/pi,'y',n,1);            % Apply elevation direction
    a.rotate_pattern( phi(n)*180/pi,'z',n,1);              % Apply azimuth direction
end
a.center_frequency = s.center_frequency;                   % Set center frequency
a.combine_pattern;                                         % Apply far field transformation
P = sum( abs(a.Fa(:,:,1:N)).^2,3 );                        % Normalize to unit power
a.Fa(:,:,1:N) = a.Fa(:,:,1:N) ./ sqrt(P(:,:,ones(1,N)));

a.Fb(:,:,N+1) = 1;                                         % Add horizontal polarization
a.Fa(:,:,N+1) = 0;

a.Fb(:,:,N+2) = 1/sqrt(2);                                 % Add LHCP receive polarization
a.Fa(:,:,N+2) = 1j/sqrt(2);
a.Fb(:,:,N+3) = 1/sqrt(2);                                 % Add RHCP receive polarization
a.Fa(:,:,N+3) = -1j/sqrt(2);
```

Layout and Channel Generation We have one transmitter and 250 receiver positions. Each receiver gets a specific channel. However, the receivers LSPs will be correlated. The BS useses a 2-element antenna that transmits a linear polarized signal and an left-hand circular polarized signal. This will allow us to verify the correct functionality for both polarizations.

```
l = qd_layout(s);                                          % Create new QuaDRiGa layout
l.no_rx = 250;                                             % Set number of MTs
l.randomize_rx_positions( 200 , 1.5 , 1.5 , 1.7 );         % 200 m radius, 1.5 m Rx height
l.set_scenario('BERLIN_UMa_NLOS');                         % Use NLOS scenario

l.tx_position(3) = 20;                                     % 20 m tx height
l.tx_array = qd_arrayant( 'omni' );                        % Omni-directional BS antenna
l.tx_array.copy_element(1,2);
l.tx_array.Fa(:,:,2) = 1/sqrt(2);                          % Send additional LHCP signal
l.tx_array.Fb(:,:,2) = 1j/sqrt(2);
l.rx_array = a;                                            % Omni-directional MT antenna

set(0,'DefaultFigurePaperSize',[14.5 7.3])                 % Adjust paper size for plot
l.visualize([],[],0);                                      % Plot the layout
view(-33, 60);                                             % Enable 3D view
```



We set up the scenario and adjust the parameter range. Then, we generate the channel coefficients. In the last step, the arrival angles are obtained from the channel coefficients. This uses only the linear polarized transmit singal.

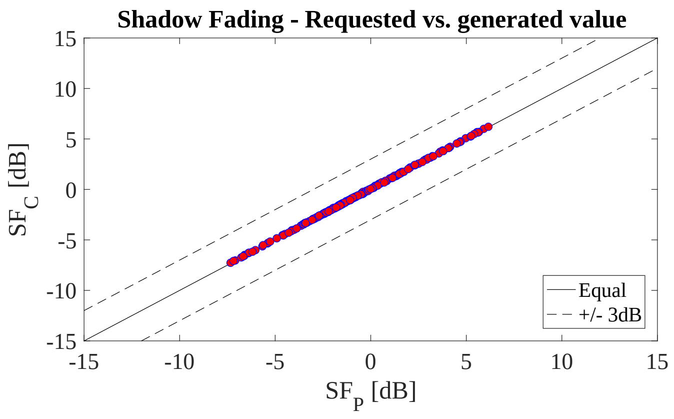
```
p = l.init_builder;                                       % Initialize builder
p.plpar = [];                                             % Disable path-loss
p.scenpar.NumClusters = 15;                               % Reduce paths (for faster processing)
p.lsp_xcorr = eye(8);                                     % Disable inter-parameter correlation

p.scenpar.XPR_mu    = 2;                                  % Set XPR range
p.scenpar.XPR_sigma = 10;
p.scenpar.KF_mu     = -5;                                 % Set KF-Range
p.scenpar.KF_sigma  = 10;
p.scenpar.DS_mu     = log10(0.6e-6);                      % Median DS = 600 ns
p.scenpar.DS_sigma  = 0.3;                                % 300-1200 ns range

p.scenpar.asA_kf = -0.6;                                  % Set some inter-parameter correlations
p.scenpar.esA_kf = -0.6;
p.scenpar.esA_asA = 0.5;

p.scenpar.PerClusterAS_A = 1;                             % Limit the per cluster AS to 1 degree
p.scenpar.PerClusterAS_D = 1;
p.scenpar.PerClusterES_A = 1;
p.scenpar.PerClusterES_D = 1;

p.gen_parameters;                                         % Generate small-scale-fading parameters
c = p.get_channels;                                       % Generate channel coefficients

coeff = cat( 5, c.coeff );                                % Extract amplitudes and phases
delay = cat( 5, c.delay );                                % Extract path delays

cf = reshape( coeff(:,1,:,:,:), a.no_elements, 1, [] ); % Format input for angle estimation
[ az, el, J ] = qf.calc_angles( cf, a, 1, [], 1 );       % Calculate angles
```

```
SSF Corr.    [oooooooooooooooooooooooooooooooooooooooooooooooooo]     2 seconds
Channels     [oooooooooooooooooooooooooooooooooooooooooooooooooo]    33 seconds
Calc Angles  [oooooooooooooooooooooooooooooooooooooooooooooooooo]   108 seconds
```

Results and discussion In the following plots, we extract parameters from the generated coefficients and compare them with the initial ones which were generated by the 'qd_builder' object (p). The values in (p) can be seen as a request to the channel builder and the values in the generated coefficients (c) as a delivery.

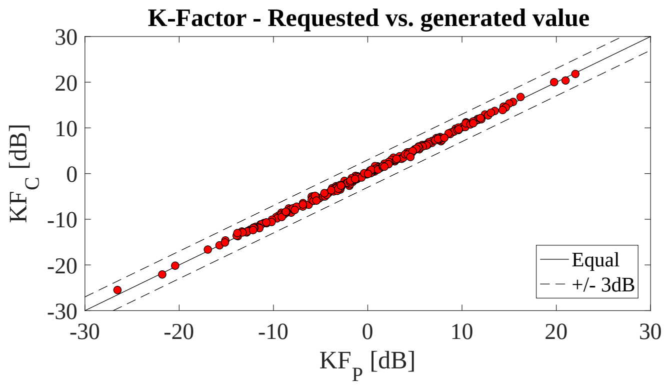
We first calculate the SF from the channel data by summing up the power over all 20 taps. We see, that the values are almost identical.

```
sf = mean(sum(sum(abs(coeff(1:29,1,:,:,:)).^2,3),1),4);     % Calculate shadow fading
sf = sf(:);

set(0,'DefaultFigurePaperSize',[14.5 4.5])                % Change Paper Size
figure('Position',[ 100 , 100 , 760 , 400]);              % New figure

plot(-35:35,-35:35,'k')
hold on
plot([-35:35]+3,-35:35,'--k')
plot([-35:35]-3,-35:35,'--k')
plot( 10*log10(p.sf) , 10*log10(sf) , 'ob','Markerfacecolor','r')
hold off
axis([ -15 , 15 , -15, 15 ])
legend('Equal','+/- 3dB','Location','SouthEast')
xlabel('SF_P [dB]'); ylabel('SF_C [dB]');
title('Shadow Fading - Requested vs. generated value');
```



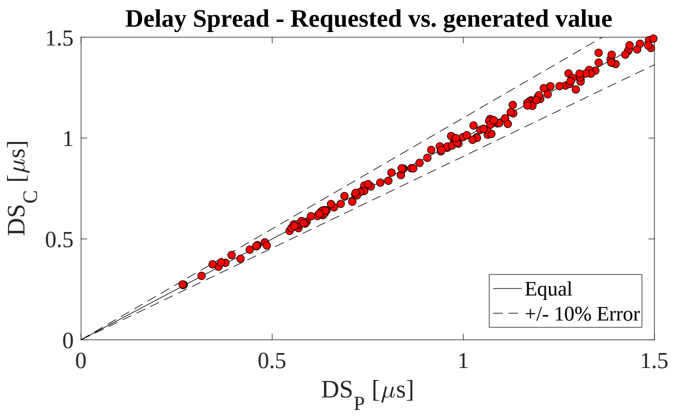
Next, we repeat the same calculation for the K-Factor. Again, we see that the values are almost identical.

```
p_nlos = mean(sum(sum(  abs( coeff(1:29,1,2:end,:,:) ).^2  ,3),1),4);      % Calculate NLOS power
p_los  = mean(sum(sum(  abs( coeff(1:29,1,  1  ,:,:) ).^2  ,3),1),4);      % Calculate LOS power
kf = p_los./p_nlos;                                       % Calculate K-Factor
kf = kf(:);

figure('Position',[ 100 , 100 , 760 , 400]);              % New figure
plot(-35:35,-35:35,'k')
hold on
plot([-35:35]+3,-35:35,'--k')
plot([-35:35]-3,-35:35,'--k')
plot( 10*log10(p.kf) , 10*log10(kf) , 'ok','Markerfacecolor','r')
hold off
axis([ -30 , 30 , -30, 30 ])
legend('Equal','+/- 3dB','Location','SouthEast')
xlabel('KF_P [dB]');
ylabel('KF_C [dB]');
title('K-Factor - Requested vs. generated value');
```



Now we repeat the calculation for the RMS delays spread.

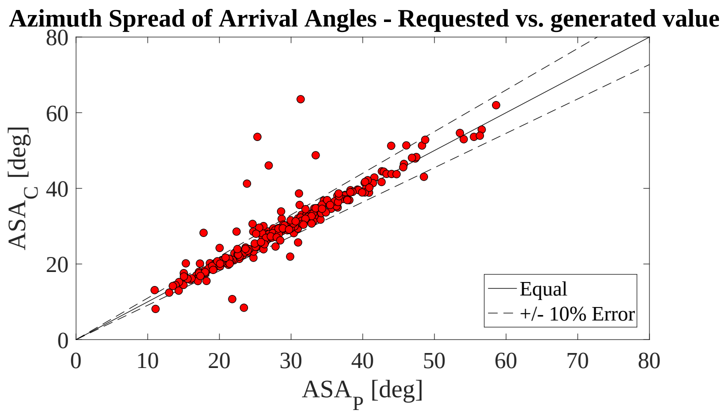
```
pow = reshape( permute( sum(abs(coeff(:,1,:,:,:)).^2,1) , [5,4,3,1,2] ), [], c(1).no_path );
tau = reshape( permute( mean( delay(:,1,:,:,:) ,1)      , [5,4,3,1,2] ), [], c(1).no_path );
pow_sum = sum(pow,2);                                     % Calculate sum-power

pow_tap = abs(coeff).^2;                                  % Calculate path powers

mean_delay = sum( pow.*tau,2) ./ pow_sum;                 % Calculate mean delay
ds = sqrt( sum( pow.*tau.^2 ,2)./ pow_sum - mean_delay.^2 );
ds = mean( reshape( ds, l.no_rx,[] ),2 );

figure('Position',[ 100 , 100 , 760 , 400]);             % New figure
plot([0:0.1:2],[0:0.1:2],'k')
hold on
plot([0:0.1:2]*1.1,[0:0.1:2],'--k')
plot([0:0.1:2],[0:0.1:2]*1.1,'--k')
plot( p.ds'*1e6 , (ds')*1e6 , 'ok','Markerfacecolor','r')
hold off
axis([ 0,1.5,0,1.5 ])
legend('Equal','+/- 10% Error','Location','SouthEast')
xlabel('DS_P [\mus]');
ylabel('DS_C [\mus]');
title('Delay Spread - Requested vs. generated value');
```



Now we compare the angular spreads calculated from the channel coefficients with the values in the builder. Most values are in the 10% error corridor. The deviations come from the per-cluster angular spreads, the limited resolution of the antenna and the dependency of the maximal angular spread on the K-Factor.

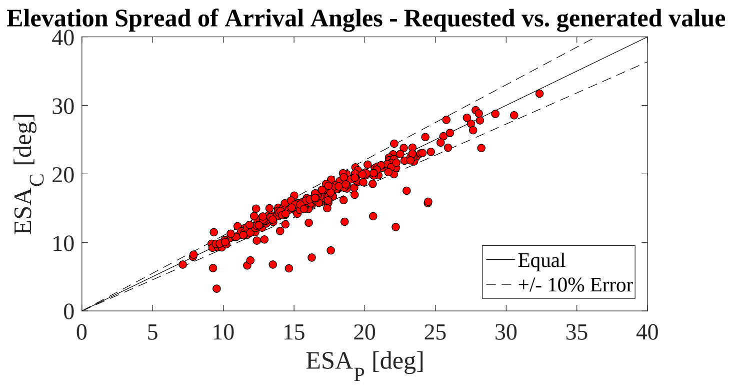
```
az = reshape( az, c(1).no_path, [], l.no_rx );

ang = reshape( permute( az, [3,2,1] ), [], c(1).no_path );
asa = qf.calc_angular_spreads( ang,pow );
asa = mean( reshape( asa, l.no_rx,[] ),2 );

```

```
figure('Position',[ 100 , 100 , 760 , 400]);            % New figure
plot([0:180],[0:180],'k')
hold on
plot([0:180]*1.1,[0:180],'--k')
plot([0:180],[0:180]*1.1,'--k')
plot( p.asA' , asa*180/pi , 'ok','Markerfacecolor','r')
hold off
axis([ 0,80,0,80 ])
legend('Equal','+/- 10% Error','Location','SouthEast')
xlabel('ASA_P [deg]');
ylabel('ASA_C [deg]');
title('Azimuth Spread of Arrival Angles - Requested vs. generated value');
```



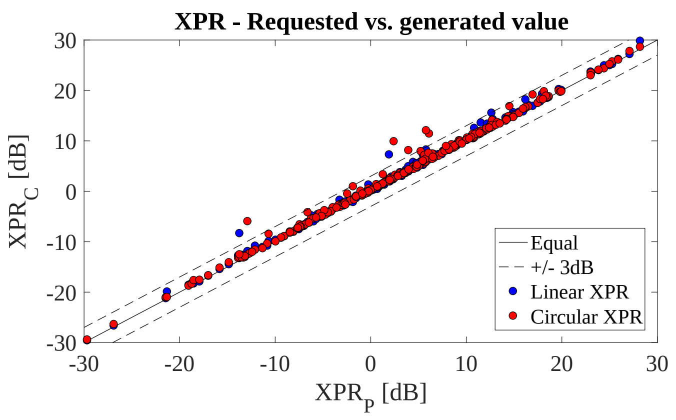
The same calculations are made for the elevation angles.

```
el = reshape( el, c(1).no_path, [], l.no_rx );

ang = reshape( permute( el, [3,2,1] ), [], c(1).no_path );
esa = qf.calc_angular_spreads( ang,pow );
esa = mean( reshape( esa, l.no_rx,[] ),2 );

figure('Position',[ 100 , 100 , 760 , 400]);            % New figure
plot([0:180],[0:180],'k')
hold on
plot([0:180]*1.1,[0:180],'--k')
plot([0:180],[0:180]*1.1,'--k')
plot( p.esA' , esa*180/pi , 'ok','Markerfacecolor','r')
hold off
axis([ 0,40,0,40 ])
legend('Equal','+/- 10% Error','Location','SouthEast')
xlabel('ESA_P [deg]');
ylabel('ESA_C [deg]');
title('Elevation Spread of Arrival Angles - Requested vs. generated value');
```



The transmitter at the BS sends a vertically polarized and a LHCP wave. When the wave is scattered, the polarization is changed. The array antenna is able to measure the Jones-vector of the incoming wave (after the reflection). Hence, it is possible to calculate the XPR of the scattering events. Likewise, the polarization of the LHCP singal is changed during scattering. We use the power-ratio of the RHCP receiv antenna to the LHCP receive antenna to determine the circular XPR.

```
xpr = abs(J(1,1,:)).^2 ./ abs(J(2,1,:)).^2;
xpr = reshape( xpr, c(1).no_path, [], l.no_rx );
xpr = mean(mean(xpr(2:end,:,:),1),2);
xpr = xpr(:);

xprC = abs(coeff(31,2,:,:,:)).^2 ./ abs(coeff(30,2,:,:,:)).^2;
xprC = mean(mean(xprC(1,1,2:end,:,:),3),4);
xprC = xprC(:);

figure('Position',[ 100 , 100 , 760 , 400]);              % New figure
plot(-35:35,-35:35,'k')
hold on
plot([-35:35]+3,-35:35,'--k')
plot( 10*log10(p.xpr)' , 10*log10(xpr) , 'ok','Markerfacecolor','b')
plot( 10*log10(p.xpr)' , 10*log10(xprC) , 'ok','Markerfacecolor','r')
plot([-35:35]-3,-35:35,'--k')
hold off
axis([ -30 , 30 , -30, 30 ])
legend('Equal','+/- 3dB','Linear XPR','Circular XPR','Location','SouthEast')
xlabel('XPR_P [dB]');
ylabel('XPR_C [dB]');
title('XPR - Requested vs. generated value');
```



Lastly, it is checked if the requested inter-parameter correlations are also found in the channel coefficients.

```
disp(['Corr. KF  - ASA :  ',num2str(qf.xcorrcoeff( 10*log10(kf) , log10(asa)    ),'%1.2f')])
disp(['Corr. KF  - ESA :  ',num2str(qf.xcorrcoeff( 10*log10(kf) , log10(esa)    ),'%1.2f')])
disp(['Corr. KF  - SF  :  ',num2str(qf.xcorrcoeff( 10*log10(kf) , 10*log10(sf) ),'%1.2f')])
disp(['Corr. KF  - DS  :  ',num2str(qf.xcorrcoeff( 10*log10(kf) , log10(ds)     ),'%1.2f')])
disp(['Corr. DS  - SF  :  ',num2str(qf.xcorrcoeff( log10(ds)    , 10*log10(sf) ),'%1.2f')])
disp(['Corr. DS  - ASA :  ',num2str(qf.xcorrcoeff( log10(ds)    , log10(asa)    ),'%1.2f')])
disp(['Corr. DS  - ESA :  ',num2str(qf.xcorrcoeff( log10(ds)    , log10(esa)    ),'%1.2f')])
disp(['Corr. ASA - ESA :  ',num2str(qf.xcorrcoeff( log10(asa)   , log10(esa)    ),'%1.2f')])
```

```
Corr. KF  - ASA :  -0.67
Corr. KF  - ESA :  -0.66
Corr. KF  - SF  :  0.22
Corr. KF  - DS  :  -0.09
Corr. DS  - SF  :  0.17
Corr. DS  - ASA :  0.19
Corr. DS  - ESA :  0.04
Corr. ASA - ESA :  0.65
```

4.8 Time Evolution and Scenario Transitions

This tutorial shows how user trajectories, segments, and scenarios are defined. Channel coefficients are created for each segment separately. The channel merger combines these output into a longer sequence.

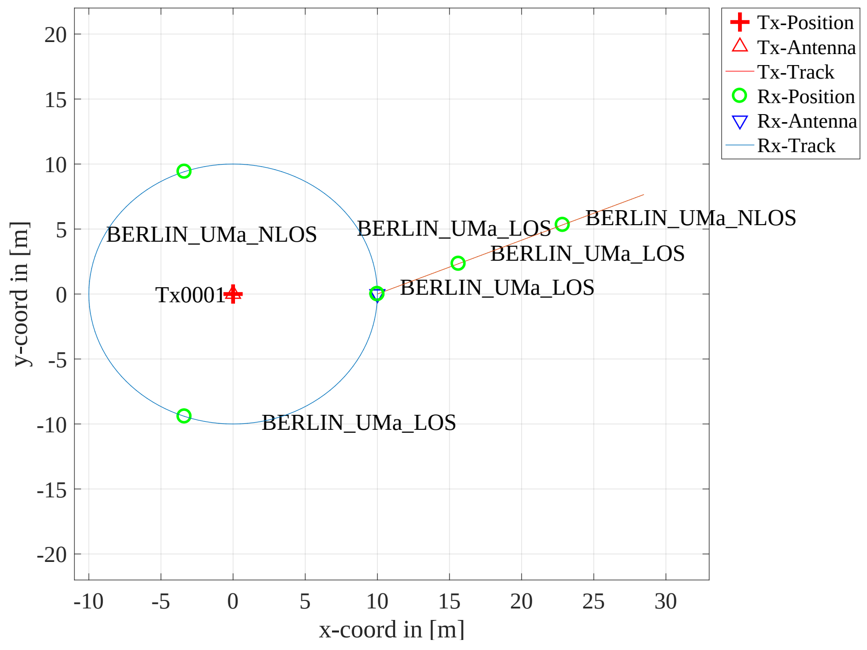
The output sequences are evaluated for different settings of the model. The channel model generates the coefficients separately for each segment. In order to get a time-continuous output, these coefficients have to be combined. This is a feature which is originally described in the documentation of the WIM2 channel model, but which was never implemented. Since this component is needed for time-continuous simulations, it was implemented here. This script sets up the simulation and creates such time-continuous CIRs.

Channel model setup and coefficient generation First, we set up the channel model.

```
close all
clear all

set(0,'defaultTextFontSize', 18)                       % Default Font Size
set(0,'defaultAxesFontSize', 18)                       % Default Font Size
set(0,'defaultAxesFontName','Times')                   % Default Font Type
set(0,'defaultTextFontName','Times')                   % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')         % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')             % Default Paper Type

s = qd_simulation_parameters;                          % New simulation parameters
s.center_frequency = 2.53e9;                           % 2.53 GHz carrier frequency
s.sample_density = 4;                                  % 4 samples per half-wavelength
s.use_absolute_delays = 1;                             % Include delay of the LOS path
s.show_progress_bars = 1;                              % Disable progress bars
```

Second, we create a more complex network layout featuring an elevated transmitter (25 m) and two receivers at 1.5 m height. The first Rx moves along a circular track around the receiver. The second receiver moves away from the Tx. Both start at the same point. Note here, that each track is split into three segments. The first Rx goes from an LOS area to a shaded area and back. The second track also start in the LOS area. Here, the scenario changes to another LOS segment and then to an NLOS segment. The LOS-LOS change will create new small-scale fading parameters, but the large scale parameters (LSPs) will be highly correlated between those two segments.

```
l = qd_layout(s);                                         % Create new QuaDRiGa layout
l.no_rx = 2;                                              % Two receivers
l.tx_array = qd_arrayant('dipole');                       % Dipole antennas at all Rx and Tx
l.rx_array = l.tx_array;
l.tx_position(3) = 25;                                    % Elevate Tx to 25 m

UMal = 'BERLIN_UMa_LOS';                                  % LOS scenario name
UMan = 'BERLIN_UMa_NLOS';                                 % NLOS scenario name

l.rx_track(1,1) = qd_track('circular',20*pi,0);           % Circular track with 10m radius
l.rx_track(1,1).initial_position  = [10;0;1.5];           % Start east, running north
l.rx_track(1,1).segment_index     = [1,40,90];            % Segments
l.rx_track(1,1).scenario          = { UMal, UMan, UMal }; % Scenarios
l.rx_track(1,1).name = 'Rx1';

l.rx_track(1,2) = qd_track('linear',20,pi/8);             % Linear track, 20 m length
l.rx_track(1,2).initial_position  = [10;0;1.5];           % Same start point
l.rx_track(1,2).interpolate_positions( 128/20 );
l.rx_track(1,2).segment_index     = [1,40,90];            % Segments
l.rx_track(1,2).scenario          = { UMal, UMal, UMan }; % Scenarios
l.rx_track(1,2).name = 'Rx2';

set(0,'DefaultFigurePaperSize',[14.5 7.3])                % Adjust paper size for plot
l.visualize;                                              % Plot the layout

interpolate_positions( l.rx_track, s.samples_per_meter ); % Interpolate
calc_orientation( l.rx_track );                           % Align antenna direction with track
```



Now we create the channel coefficients. The fixing the random seed guarantees repeatable results (i.e. the taps will be at the same positions for both runs). Also note the significantly longer computing time when drifting is enabled.

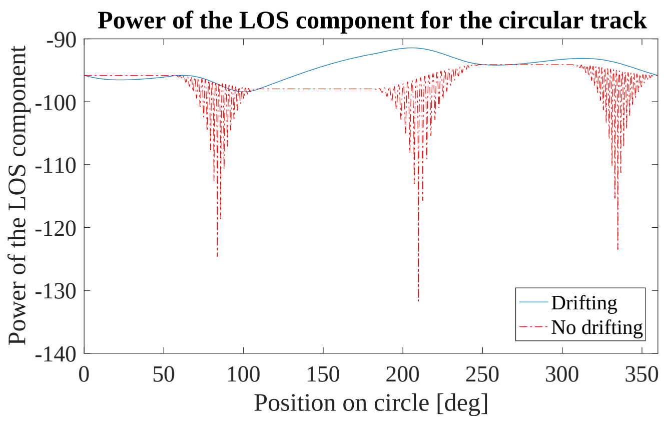
```
disp('Drifting enabled:');
p = l.init_builder;                                        % Create channel builders
gen_parameters( p );                                       % Generate small-scale fading
c = get_channels( p );                                     % Generate channel coefficients
cn = merge( c );

disp('Drifting disabled:');
warning('off','QuaDRiGa:qd_builder:gen_ssf_parameters:exisitng')

s.use_3GPP_baseline = 1;                                   % Disable drifting
gen_parameters(p,2);                                       % Update small-scale fading
gen_parameters(p,3);                                       % Calc. FBS / LBS Positions
d = get_channels( p );                                     % Generate channel coefficients
dn = merge( d );
```

```
Drifting enabled:
SSF Corr.     [oooooooooooooooooooooooooooooooooooooooooooooooooo]     1 seconds
Channels      [oooooooooooooooooooooooooooooooooooooooooooooooooo]    40 seconds
Merging       [oooooooooooooooooooooooooooooooooooooooooooooooooo]     1 seconds
Drifting disabled:
SSF Corr.     [oooooooooooooooooooooooooooooooooooooooooooooooooo]     0 seconds
SSF Corr.     [oooooooooooooooooooooooooooooooooooooooooooooooooo]     0 seconds
Channels      [oooooooooooooooooooooooooooooooooooooooooooooooooo]     3 seconds
Merging       [oooooooooooooooooooooooooooooooooooooooooooooooooo]     0 seconds
```

Results and discussion Now we plot the and discuss the results. We start with the power of the LOS tap along the circular track and compare the outcome with and without drifting.

```
degrees = (0:cn(1,1).no_snap-1)/cn(1).no_snap * 360;
los_pwr_drift = 10*log10(squeeze(abs(cn(1).coeff(1,1,1,:))).^2);
los_pwr_nodrift = 10*log10(squeeze(abs(dn(1).coeff(1,1,1,:))).^2);

set(0,'DefaultFigurePaperSize',[14.5 4.5])             % Change Paper Size
figure('Position',[ 100 , 100 , 760 , 400]);           % New figure
plot( degrees,los_pwr_drift )
hold on
plot(degrees,los_pwr_nodrift ,'-.r')
hold off

```

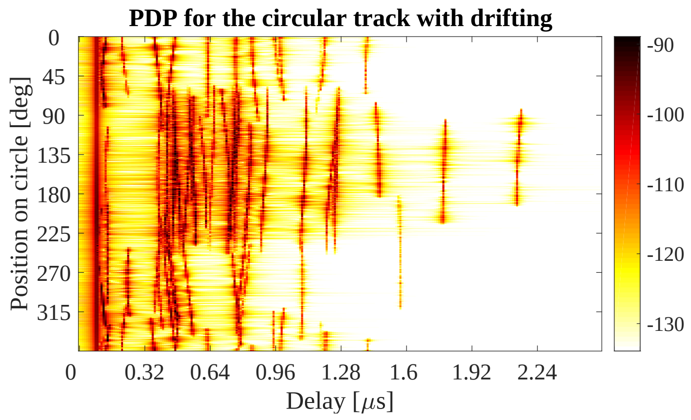
```
a = axis; axis( [0 360 a(3:4) ] );
xlabel('Position on circle [deg]');
ylabel('Power of the LOS component');
title('Power of the LOS component for the circular track');
legend('Drifting','No drifting','Location','SouthEast');
```



When drifting is enabled (blue curve), the channel output after merging is time-continuous. The variations along the track come from the drifting K-Factor and the drifting shadow fading. When drifting is disabled, these parameters are not updated and kept fixed at their initial value. At the end of each segment, both channels are cross-faded, i.e. the power of the output of the first segment ramps down and the power of the second segment ramps up. Since drifting guarantees a time-continuous evolution of the phase, this ramping process is also time continuous and no artifacts are visible in the blue curve. Without drifting, the phases are approximated based on their initial values, the initial arrival and departure angles and the traveled distance from the start point. However, since the Rx moves along a circular track, the angles change continuously which is not correctly modeled. The phase at the end of the first segment does not match the phase at the beginning of the second. When adding both components, artifacts appear as can be seen in the red curve.

Next, we plot the power-delay profiles for both tracks. We calculate the frequency response of the channel and transform it back to time domain by an IFFT. Then, we create a 2D image of the received power at each position of the track. We start with the circular track.

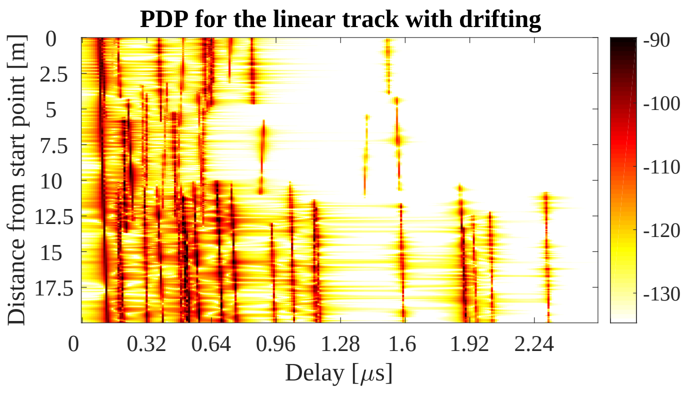
```
h = cn(1,1).fr( 100e6,512 );                            % Freq.-domain channel
h = squeeze(h);                                         % Remove singleton dimensions
pdp = 10*log10(abs(ifft(h,[],1).').^2);                 % Power-delay profile

figure('Position',[ 100 , 100 , 760 , 400]);            % New figure
imagesc(pdp(:,1:256));

caxis([ max(max(pdp))-50 max(max(pdp))-5 ]); colorbar;  % Figure decorations
cm = colormap('hot'); colormap(cm(end:-1:1,:));
set(gca,'XTick',1:32:255); set(gca,'XTickLabel',(0:32:256)/100e6*1e6);
set(gca,'YTick',1:cn(1).no_snap/8:cn(1).no_snap);
set(gca,'YTickLabel', (0:cn(1).no_snap/8:cn(1).no_snap)/cn(1).no_snap * 360 );
xlabel('Delay [\mus]'); ylabel('Position on circle [deg]');
title('PDP for the circular track with drifting');
```



The X-axis shows the delay in microseconds and the Y-axis shows the position on the circle. For easier navigation, the position is given in degrees. 0 deg means east (starting point), 90 deg means north, 180 deg west and 270 deg south. The LOS delay stays constant since the distance to the Tx is also constant. However, the power of the LOS changes according to the scenario. Also note, that the NLOS segment has more paths due to the longer delay spread.

Next, we create the same plot for the linear track. Note the slight increase in the LOS delay and the high similarity of the first two LOS segments due to the correlated LSPs. Segment change is at around 6 m.

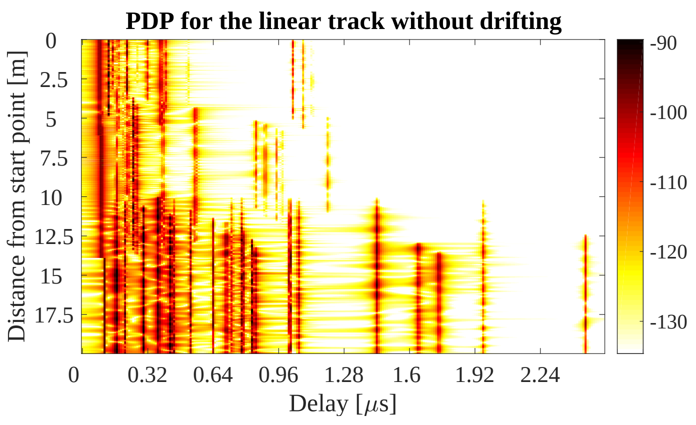
```
h = cn(1,2).fr( 100e6,512 );                              % Freq.-domain channel
h = squeeze(h);                                            % Remove singleton dimensions
pdp = 10*log10(abs(ifft(h,[],1).').^2);                    % Power-delay profile

figure('Position',[ 100 , 100 , 760 , 400]);               % New figure
imagesc(pdp(:,1:256));

caxis([ max(max(pdp))-50 max(max(pdp))-5 ]); colorbar;  % Figure decorations
cm = colormap('hot'); colormap(cm(end:-1:1,:));
set(gca,'XTick',1:32:255); set(gca,'XTickLabel',(0:32:256)/100e6*1e6);
set(gca,'YTick',1:cn(2).no_snap/8:cn(2).no_snap);
set(gca,'YTickLabel', (0:cn(2).no_snap/8:cn(2).no_snap)/cn(2).no_snap * 20 );
xlabel('Delay [\mus]'); ylabel('Distance from start point [m]');
title('PDP for the linear track with drifting');
```



Last, we plot the same results for the linear track without drifting. Note here, that the LOS delay is not smooth during segment change. There are two jumps at 6 m and again at 13.5 m.

```
h = dn(1,2).fr( 100e6,512 );                              % Freq.-domain channel
h = squeeze(h);                                            % Remove singleton dimensions
pdp = 10*log10(abs(ifft(h,[],1).').^2);                    % Power-delay profile

figure('Position',[ 100 , 100 , 760 , 400]);               % New figure
imagesc(pdp(:,1:256));
```

```

caxis([ max(max(pdp))-50 max(max(pdp))-5 ]); colorbar;  % Figure decorations
cm = colormap('hot'); colormap(cm(end:-1:1,:));
set(gca,'XTick',1:32:255); set(gca,'XTickLabel',(0:32:256)/100e6*1e6);
set(gca,'YTick',1:cn(2).no_snap/8:cn(2).no_snap);
set(gca,'YTickLabel', (0:cn(2).no_snap/8:cn(2).no_snap)/cn(2).no_snap * 20 );
xlabel('Delay [\mus]'); ylabel('Distance from start point [m]');
title('PDP for the linear track without drifting');
```



4.9 Applying Varying Speeds (Channel Interpolation)

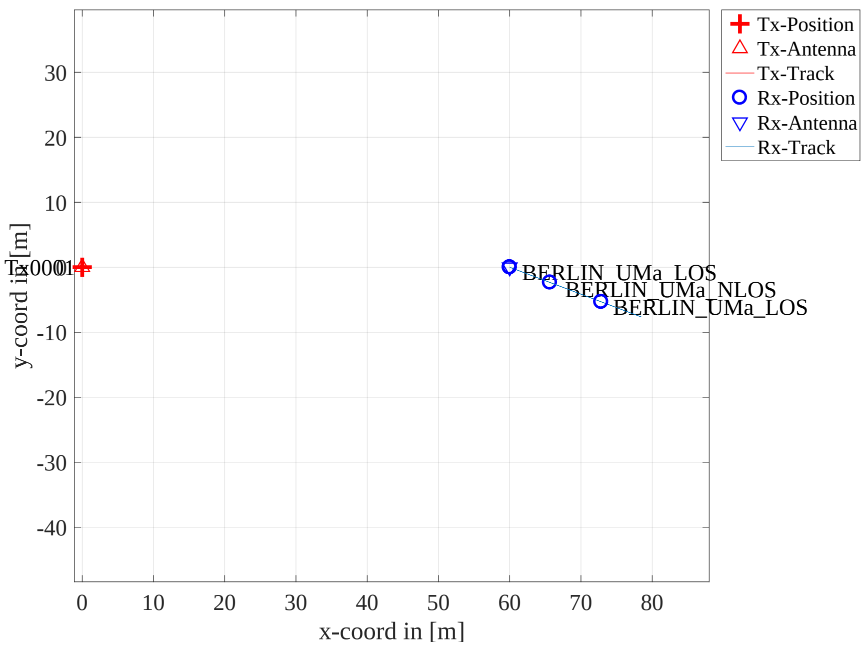
This tutorial shows how to adjust the speed of the terminal, e.g. when breaking or accelerating. First, a simple scenario defined. Channel coefficients are calculated at a constant speed and then interpolated to match the varying speed of the terminal. One feature that makes the simulations more realistic is the function to apply arbitrary speed- and movement profiles, e.g. accelerating, breaking or moving at any chosen speed. These profiles are defined in the track class. The profiles are then converted in to effective sampling points which aid the interpolation of the channel coefficients.

Channel model set-up First, we set up the simulation parameters. Note the sample density of 1.2 which enables very fast simulations even with drifting. The sample density must fulfill the Nyquist theorem, i.e., there must be at least 1 sample per half-wavelength in order to be able to interpolate the channels correctly. Note that when both transmitter and receiver are mobile, the minimum value is 2 since they may move towards each other.

```
close all
clear all

set(0,'defaultTextFontSize', 18)                        % Default Font Size
set(0,'defaultAxesFontSize', 18)                        % Default Font Size
set(0,'defaultAxesFontName','Times')                    % Default Font Type
set(0,'defaultTextFontName','Times')                    % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')          % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')              % Default Paper Type

s = qd_simulation_parameters;                           % New simulation parameters
s.center_frequency = 2.53e9;                            % 2.53 GHz carrier frequency
s.sample_density = 1.2;                                 % 2.5 samples per half-wavelength
s.use_absolute_delays = 1;                              % Include delay of the LOS path
s.show_progress_bars = 0;                               % Disable progress bars
```

Second, we define a track. It has a length of 20 m, starts at 10 m east of the transmitter and consists of three segments (LOS, NLOS, LOS). The positions are interpolated to match the sample density defined above. The track is then plugged into a network layout with one transmitter at position (0,0,25). Both, transmitter and receiver are equipped with dipole antennas. The last three lines create the large scale parameters (LSPs).

```
t = qd_track('linear',20,-pi/8);                        % 20 m track, direction SE
t.initial_position = [60;0;1.5];                        % Start position
t.interpolate_positions( 128/20 );                      % Interpolate
t.segment_index       = [1,40,90];                      % Assign segments
t.scenario            = {'BERLIN_UMa_LOS','BERLIN_UMa_NLOS','BERLIN_UMa_LOS'};
t.interpolate_positions( s.samples_per_meter );         % Apply sample density

l = qd_layout( s );                                     % New QuaDRiGa layout
l.tx_array = qd_arrayant('dipole');                     % Set Dipole antenna
l.rx_array = qd_arrayant('dipole');                     % Set Dipole antenna
l.tx_position(3) = 25;                                  % BE height
l.rx_track = t;                                         % Assign track

set(0,'DefaultFigurePaperSize',[14.5 7.3])              % Adjust paper size for plot
l.visualize;                                            % Plot the layout
```



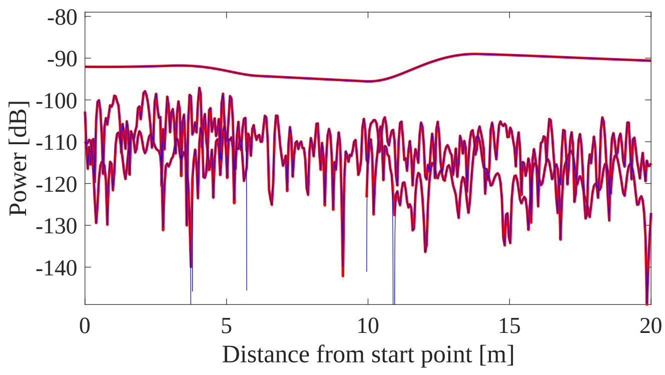
Channel generation and results Next, we generate the channel coefficients. Note that here, the initial sample density is 1.2. We then interpolate the sample density to 20. It would take ten times as long to achieve the same result with setting the initial sample density to 20. The interpolation is significantly faster. It is done by first setting the speed to 1 m/s (default setting) and then creating a distance vector which contains a list of effective sampling points along the track.

```
cn = l.get_channels;                                    % Generate channels

t.set_speed( 1 );                                       % Set constant speed
dist = t.interpolate_movement( s.wavelength/(2*20) );   % Get snapshot positions
ci = cn.interpolate( dist );                            % Interpolate channels
```

The next plot shows the power of the first three taps from both, the original and the interpolated channel, plotted on top of each other. The values are identical except for the fact, that the interpolated values (blue line) have 17 times as many sample points.

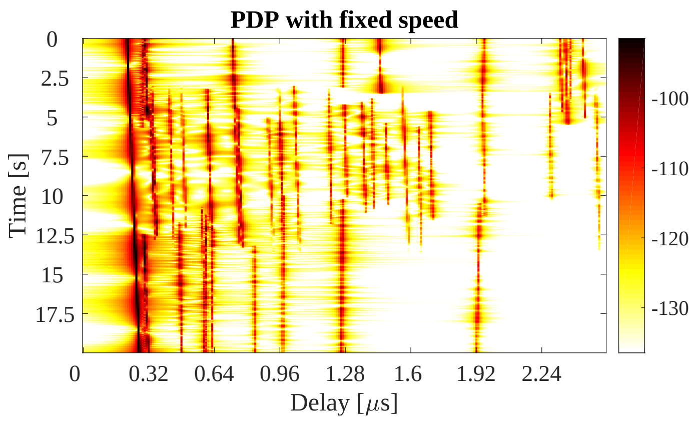
```
nsnap = cn.no_snap;                                     % No. snapshots
dist_orig = (0:nsnap-1) * t.get_length/(nsnap-1);       % Distances
pwr_orig  = 10*log10(squeeze(abs(cn.coeff(1,1,1:3,:))).^2);  % Power before interpolation
pwr_int   = 10*log10(squeeze(abs(ci.coeff(1,1,1:3,:))).^2);  % Power after interpolation

set(0,'DefaultFigurePaperSize',[14.5 4.5])              % Change Paper Size
figure('Position',[ 100 , 100 , 760 , 400]);            % New figure

plot( dist_orig,pwr_orig , 'r','Linewidth',2 )
```

```
hold on
plot( dist,pwr_int ,'b' )
hold off
axis([min(dist),max(dist), min( pwr_orig( pwr_orig>-160 ) ),...
    max( pwr_orig( pwr_orig>-160 ) )+10 ] );
xlabel('Distance from start point [m]'); ylabel('Power [dB]');
```



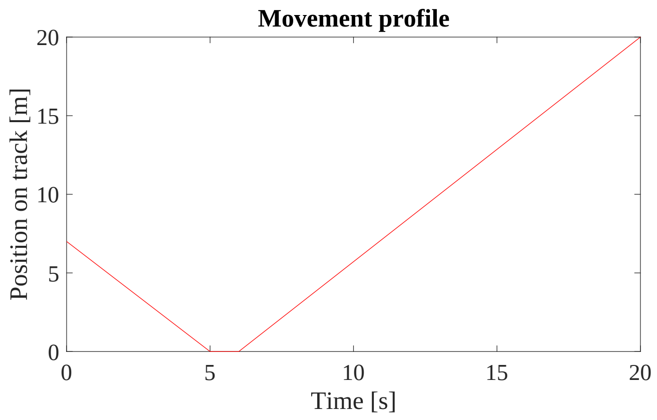
The following plot shows the power delay profile (PDP) for the interpolated channel. As defined in the track object, it starts with a LOS segment, going into a shaded area with significantly more multipath fading at around 4 seconds and then back to LOS at around 13 sec.

```
h = ci.fr( 100e6,512 );                                    % Freq.-domain channel
h = squeeze(h);                                            % Remove singleton dimensions
pdp = 10*log10(abs(ifft(h,[],1).').^2);                    % Power-delay profile

figure('Position',[ 100 , 100 , 760 , 400]);               % New figure
imagesc(pdp(:,1:256));

caxis([ max(max(pdp))-50 max(max(pdp))-5 ]); colorbar;   % Figure decorations
cm = colormap('hot'); colormap(cm(end:-1:1,:));
set(gca,'XTick',1:32:255); set(gca,'XTickLabel',(0:32:256)/100e6*1e6);
set(gca,'YTick',1:ci.no_snap/8:ci.no_snap);
set(gca,'YTickLabel', (0:ci.no_snap/8:ci.no_snap)/ci.no_snap * 20 );
xlabel('Delay [\mus]'); ylabel('Time [s]');
title('PDP with fixed speed');
```



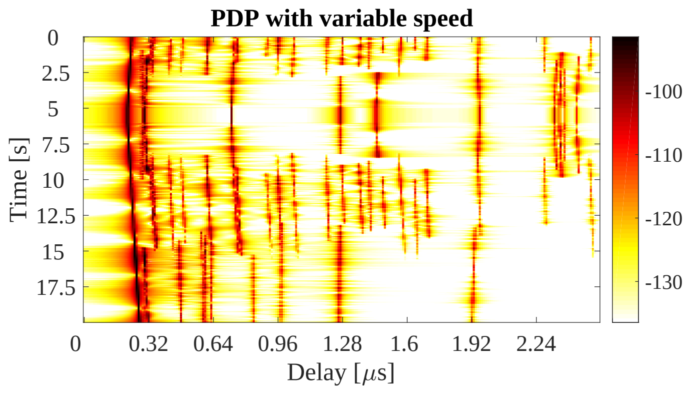
Now, we create a movement profile. It is defined by a set of value pairs in track.movement_profile. The first value represents the time in seconds, the second value the position on the track. Here, we start at a position of 7 m, i.e. in the second (NLOS) segment. We then go back to the beginning of the track. This takes 5 seconds. Then, we wait there for 1 second and go to the end of the track, which we reach after additional 14 seconds. The next step is to interpolate the sample points. This is done by the interpolate_movement method. It requires the sample interval (in s) as an input argument. Here, we choose an interval of 1 ms which gives us 1000 samples per second. The plot the illustrates the results.

```
t.movement_profile = [ 0,7 ; 5,0 ; 6,0 ; 20,20  ]';      % Generate movement profile
dist = t.interpolate_movement( 1e-3 );                   % Get snapshot positions
ci = cn.interpolate( dist );                             % Interpolate channels

nsnap = ci.no_snap;
time = (0:nsnap-1) * t.movement_profile(1,end)/(nsnap-1);

figure('Position',[ 100 , 100 , 760 , 400]);             % New figure
plot( time,dist , 'r' )
xlabel('Time [s]'); ylabel('Position on track [m]');
title('Movement profile');
```



The last plot shows the PDP of the interpolated channel with the movement profile applied. The channel starts in the second segment with a lot of fading, goes back to the first while slowing down at the same time. After staying constant for one second, the channel starts running again, speeding up towards the end of the track.

```
h = ci.fr( 100e6,512 );                                  % Freq.-domain channel
h = squeeze(h);                                          % Remove singleton dimensions
pdp = 10*log10(abs(ifft(h,[],1).').^2);                  % Power-delay profile

figure('Position',[ 100 , 100 , 760 , 400]);             % New figure
imagesc(pdp(:,1:256));

caxis([ max(max(pdp))-50 max(max(pdp))-5 ]); colorbar;   % Figure decorations
cm = colormap('hot'); colormap(cm(end:-1:1,:));
set(gca,'XTick',1:32:255); set(gca,'XTickLabel',(0:32:256)/100e6*1e6);
set(gca,'YTick',1:ci.no_snap/8:ci.no_snap);
set(gca,'YTickLabel', (0:ci.no_snap/8:ci.no_snap)/ci.no_snap * 20 );
xlabel('Delay [\mus]'); ylabel('Time [s]');
title('PDP with variable speed');
```



The following code segment shows a movie of the channel response. (You nedd to run the code manually in MATLAB or Octave)

```
if 0
    h = ci.fr( 20e6,128 );
    h = squeeze(h);
    mi = -90; ma = -80;
    while true
        for n = 1:size(h,2)
            pdp  = 10*log10(abs(h(:,n)).^2);
            plot(pdp)
            ma = max( ma,max([pdp]) );
            mi = min( mi,min([pdp]) );
            axis([1,128,mi,ma])
            title(round(time(n)))
            drawnow
        end
    end
end
```

4.10 Resimulating a Measured Scenario

This more complex tutorial shows how to manually define a state sequence (i.e. a sequence of scenario transitions), manipulate antennas, create large-scale-parameters such as shadow fading and delay spread, and obtain a time series of channel coefficients. This script recreates a measured drive test from the Park Inn Hotel at Berlin Alexanderplatz. The transmitter was at the rooftop of the hotel while the mobile receiver was moving south on Grunerstraße. A simplified version of the scenario is recreated in the simulation where the scenarios along the track were classified by hand.

Channel model set-up and coefficient generation The following code configures some basic parameters.

```
close all
clear all

set(0,'defaultTextFontSize', 18)                      % Default Font Size
set(0,'defaultAxesFontSize', 18)                      % Default Font Size
set(0,'defaultAxesFontName','Times')                  % Default Font Type
set(0,'defaultTextFontName','Times')                  % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')        % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')            % Default Paper Type

s = qd_simulation_parameters;                         % New simulation parameters
s.center_frequency = 2.53e9;                          % 2.53 GHz carrier frequency
s.use_absolute_delays = 1;                            % Include delay of the LOS path
s.show_progress_bars = 0;                             % Disable progress bars
```

We generate a track of 500 m length. This track is then interpolated to 1 snapshot per meter. In this way, it is possible to assign segments to the track using units of meters. The "segment_index" contains the segment start points in units of meters relative to the track start point.

```
t = qd_track('linear',500,-135*pi/180);               % Track of 500 m length, direction SE
t.initial_position = [120;-120;0];                    % Start position
t.interpolate_positions( 1 );                         % Interpolate to 1 sample per meter
t.segment_index = [1,45,97,108,110,160,190,215,235,245,280,295,304,330,400,430 ]; % Segments
```

We now assign the the scenarios to the segments. Since the measurements were done in a satellite context, we use the "MIMOSA_10-45_LOS" and "MIMOSA_10-45_NLOS" scenario. The track is then interpolated to 3 snapshots per meter.

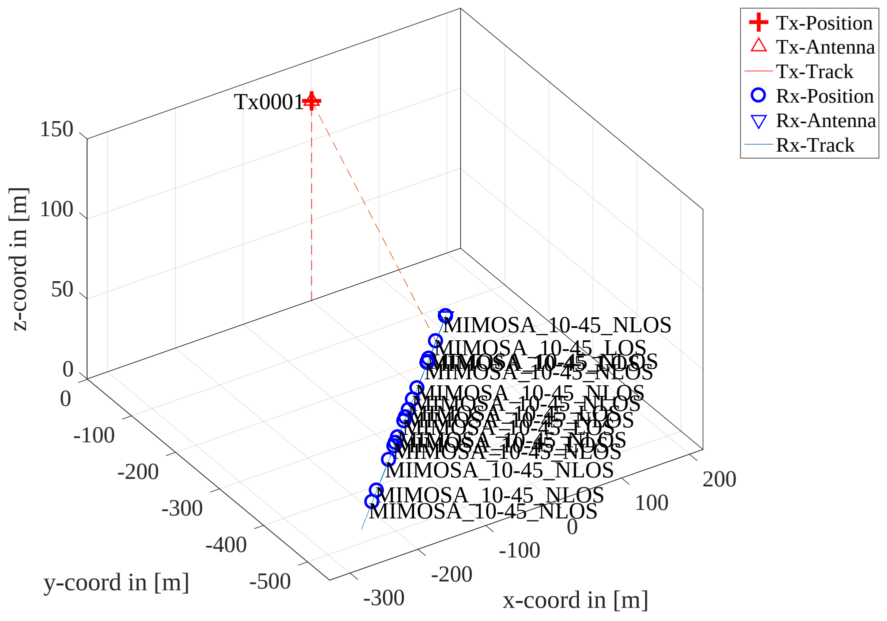
```
Sl = 'MIMOSA_10-45_LOS';
Sn = 'MIMOSA_10-45_NLOS';
t.scenario = {Sn,Sl,Sn,Sl,Sn,Sn,Sn,Sl,Sn,Sl,Sn,Sl,Sn,Sn,Sn,Sn};
t.interpolate_positions( 3 );                         % Interpolate to 3 sample per meter
```

A new QuaDRiGa layout is created, simulations parameters and the receiver track get assigned. When the channel coefficients are generated, there is a merging interval at the end of each segment during which paths from the old segment disappear and new paths appear. The method "correct_overlap" adjusts the segment start and end-points such that this transitions happens in the middle of the assigned segment start and end-points.

```
l = qd_layout( s );                                    % New QuaDRiGa layout
l.tx_position = [0;0;125];                             % Set the position of the Tx
l.rx_track = copy( t );                                % Set the rx-track
l.rx_track.correct_overlap;                            % Adjust state change position
```

Now, we assign antennas and set the antenna orientations.

```
l.tx_array = qd_arrayant('lhcp-rhcp-dipole');          % Generate Tx antenna
l.tx_array.rotate_pattern(30,'y');                     % 30 deg downtilt
l.tx_array.rotate_pattern(-90,'z');                    % point southwards

l.rx_array = qd_arrayant('lhcp-rhcp-dipole');          % Rx-Antenna
l.rx_array.rotate_pattern(-90,'y');                    % point skywards

set(0,'DefaultFigurePaperSize',[14.5 7.3])             % Adjust paper size for plot
l.visualize;                                           % Plot the layout
view(-33, 45);                                         % 3D view

% Plot a line from the Tx to the Rx
lnk = [ l.tx_position,...
    l.rx_track.positions(:,l.rx_track.segment_index(2)) + l.rx_track.initial_position ];
hold on; plot3( lnk(1,:),lnk(2,:),lnk(3,:) , '--' ); hold off
```



The last step generates the channel coefficients. The warning 'Sample density in tracks does not fulfill the sampling theoreme.' refers to the violation of the Nyquist theoreme. Hence, it will not be possible to interpolate the channel coefficients. However, this is not required here since we are only interested in the channel statistics.

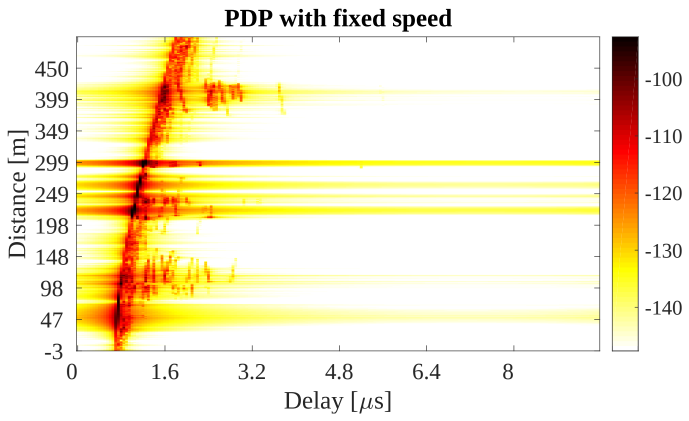
```
cn = l.get_channels;                                   % Generate channel coefficients
cn.individual_delays = 0;                              % Remove per-antenna delays
```

```
Warning: Sample density in tracks does not fulfill the sampling theoreme.
```

Results First, we plot the PDP vs. distance from the start point. For this, the channel bandwidth is reduced to 20 MHz. You can see how the delay of the LOS path sifts with the distance between BS and MT, how the LOS segments have more power, and how NLOS paths appear and disappear along the track.

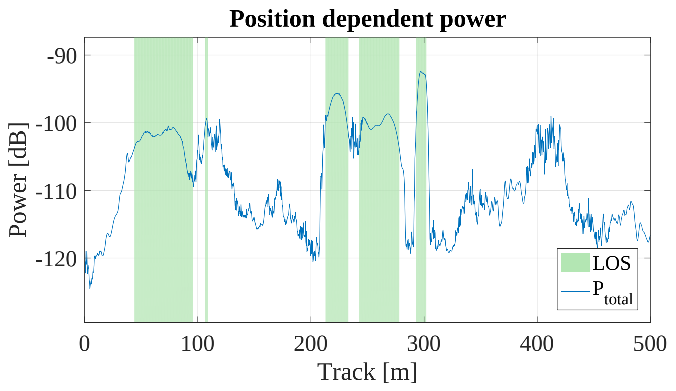
```
h =  cn.fr( 20e6,256 );                                % Freq.-domain channel
pdp = squeeze(sum(sum( abs(ifft(h,[],3)).^2 , 1),2));
pdp = 10*log10(pdp.');
```

```

set(0,'DefaultFigurePaperSize',[14.5 4.5])                  % Change paper Size
figure('Position',[ 100 , 100 , 760 , 400]);                % New figure
imagesc(pdp(end:-1:1,1:192));

caxis([ max(max(pdp))-60 max(max(pdp))-5 ]); colorbar;   % Figure decorations
cm = colormap('hot'); colormap(cm(end:-1:1,:));
set(gca,'XTick',1:32:192); set(gca,'XTickLabel',(0:32:192)/20e6*1e6);
ind = sort(cn.no_snap : -cn.no_snap/10 : 1 );
set(gca,'YTick', ind );
set(gca,'YTickLabel', round(sort(500-ind / 3,'descend')) );
xlabel('Delay [\mus]'); ylabel('Distance [m]');
title('PDP with fixed speed');
```



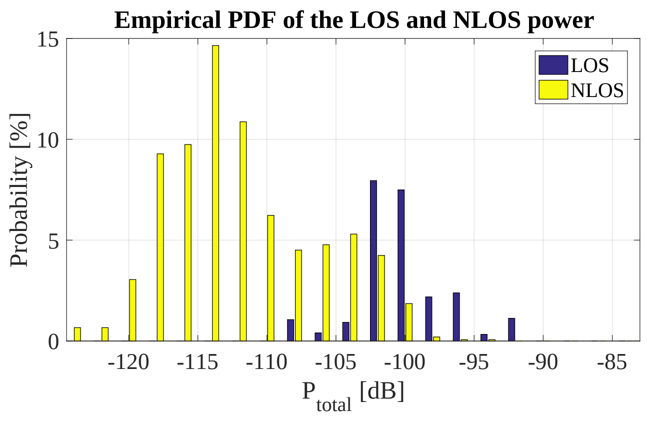
The next plot shows the total received power along the trajectory. Green shaded ares are LOS. The rest is NLOS. You can see that there is more power when there is LOS propagation.

```
dist = (1:cn.no_snap)*t.get_length/cn.no_snap;              % Traveled distance
ind  = find(strcmp(t.scenario,S1));                        % Find LOS scenarios
los  = [];
for n = 1:numel(ind)
    los = [los t.segment_index(ind(n)) : t.segment_index(ind(n)+1)];
end
ar = zeros(1,cn.no_snap); ar(los) = -200;

power = 10*log10( sum( reshape( abs(cn.coeff).^2 , [] , cn.no_snap ) ,1)/4 );

figure('Position',[ 100 , 100 , 760 , 400]);                % New figure
a = area(dist,ar);                                          % Shading for the LOS
set(a(1),'FaceColor',[0.7 0.9 0.7]); set(a,'LineStyle','none');
hold on; plot(dist,power); hold off                         % Plot the received power
title('Position dependent power'); xlabel('Track [m]'); ylabel('Power [dB]');
axis([0 500 min(power)-5 max(power)+5]); grid on;
legend('LOS','P_{total}','Location','SouthEast')
```



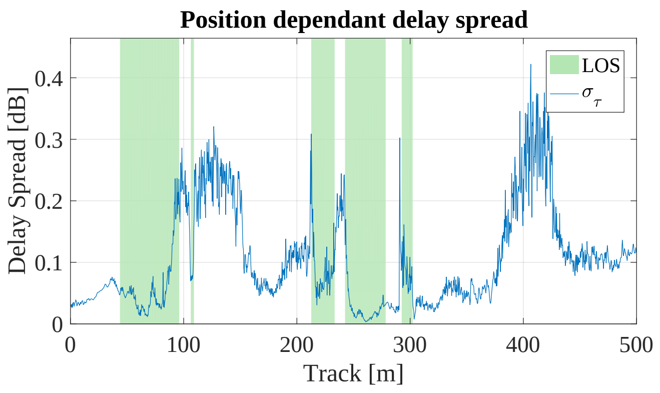
The following plot shows the distribution (PDF) of the received power for both, the LOS and NLOS segments.

```
bins   = -150:2:-80;
p_los  = hist(power(los),bins)/cn.no_snap*100;
p_nlos = hist(power(setdiff(1:cn.no_snap,los)),bins)/cn.no_snap*100;

figure('Position',[ 100 , 100 , 760 , 400]);              % New figure
bar(bins,[p_los;p_nlos]');
axis([-124.5,-83,0,ceil(max([p_los,p_nlos]))]); grid on
title('Empirical PDF of the LOS and NLOS power')
xlabel('P_{total} [dB]'); ylabel('Probability [%]'); legend('LOS','NLOS')
```



The next plot shows the RMS delay spread along the path. Again, shaded ares are for the LOS segments. Due to the strong LOS component, the DS gets shorter during LOS areas.

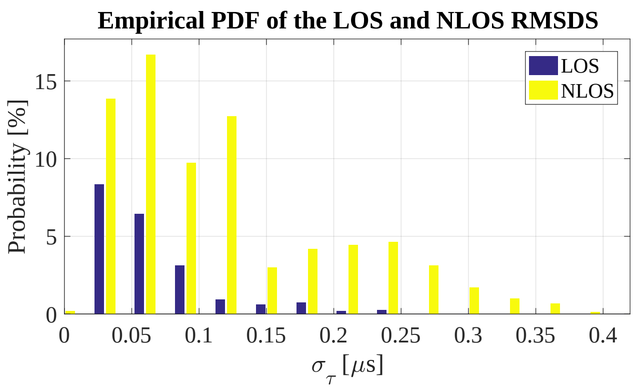
```
pow_tap = squeeze(sum(sum(abs(cn.coeff).^2,1),2));
pow_sum = sum( pow_tap,1 );
mean_delay = sum( pow_tap.*cn.delay ,1) ./ pow_sum;
ds = sqrt( sum( pow_tap.*cn.delay.^2 ,1)./ pow_sum - mean_delay.^2 );
ar = zeros(1,cn.no_snap);
ar(los) = 10;

figure('Position',[ 100 , 100 , 760 , 400]);              % New figure
a = area(dist,ar);
set(a(1),'FaceColor',[0.7 0.9 0.7]);set(a,'LineStyle','none')
hold on; plot( dist , ds*1e6 ); hold off;                 % Plot DS
ma = 1e6*( max(ds)+0.1*max(ds) );axis([0 500 0 ma]);
title('Position dependant delay spread'); grid on
xlabel('Track [m]'); ylabel('Delay Spread [dB]'); legend('LOS','\sigma_\tau');
```



The following plot shows the distribution (PDF) of the RMS delay spread for both, the LOS and NLOS segments.

```
bins = 0:0.03:3;
ds_los  = hist(ds(los)*1e6,bins)/cn.no_snap*100;
```

```
ds_nlos = hist(ds(setdiff(1:cn.no_snap,los))*1e6,bins)/cn.no_snap*100;

DS  = [ ds_los ; ds_nlos ];
ind = max( find( max(DS/max(DS(:)))>0.001 ) );

figure('Position',[ 100 , 100 , 760 , 400]);              % New figure
bar(bins,DS');
axis([0,bins(ind),0,max(DS(:))+1]); grid on;
title('Empirical PDF of the LOS and NLOS RMSDS')
xlabel('\sigma_\tau [\mus]'); ylabel('Probability [%]'); legend('LOS','NLOS');
```



4.11 Multi-frequency simulations

This tutorial demonstrates how to perform simultaneous multi-frequency simulations at two carrier frequencies: 2.6 GHz and 28 GHz in an Urban-Macrocell deployment. The BS is equipped with two different array antennas. A conventional high-gain antenna operates at 2.6 GHz. The higher frequency band uses a massive-MIMO array antenna with in an 8x8 dual-polarized setup. The model is consistent in both, the spatial domain and the frequency domain. Simulation assumptions are in accordance with 3GPP 38.901 v14.1.0 (see Section 7.6.5 Correlation modeling for multi-frequency simulations).

Identical parameters for each frequency:

- LOS / NLOS state must be the same
- BS and MT positions are the same (antenna element positions are different!)
- Cluster delays and angles for each multi-path component are the same
- Spatial consistency of the LSPs is identical

Differences:

- Antenna patterns are different for each frequency
- Path-loss is different for each frequency
- Path-powers are different for each frequency
- Delay- and angular spreads are different
- K-Factor is different
- XPR of the NLOS components is different

Basic setup Multiple frequencies are set in the simulation parameters by providing a vector of frequency sample points. A new layout is created with one 25 m high BS positions and 100 MT positions. The MTs are placed in accordance with the 3GPP assumptions, where 80% of them are situated indoors at different floor levels.

```
close all
clear all

set(0,'defaultTextFontSize', 18)                        % Default Font Size
set(0,'defaultAxesFontSize', 18)                        % Default Font Size
set(0,'defaultAxesFontName','Times')                    % Default Font Type
set(0,'defaultTextFontName','Times')                    % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')          % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')              % Default Paper Type
set(0,'DefaultFigurePaperSize',[14.5 7.3])              % Default Paper Size

s = qd_simulation_parameters;
s.center_frequency = [2.6e9, 28e9];                     % Assign two frequencies

l = qd_layout( s );                                     % New QuaDRiGa layout
l.tx_position = [0 0 25]';                              % 25 m BS height
l.no_rx = 100;                                          % 100 MTs

l.randomize_rx_positions( 200, 1.5, 1.5, 0 );           % Assign random user positions
l.rx_position(1,:) = l.rx_position(1,:) + 220;          % Place users east of the BS

floor = randi(5,1,l.no_rx) + 3;                         % Set random floor levels
for n = 1:l.no_rx
    floor( n ) =  randi(  floor( n ) );
end
l.rx_position(3,:) = 3*(floor-1) + 1.5;

indoor_rx = l.set_scenario('3GPP_38.901_UMa',[],[],0.8);    % Set the scenario
l.rx_position(3,~indoor_rx) = 1.5;                      % Set outdoor-users to 1.5 m height
```

```
Setting RX LOS state correlation distance to 50 m
```

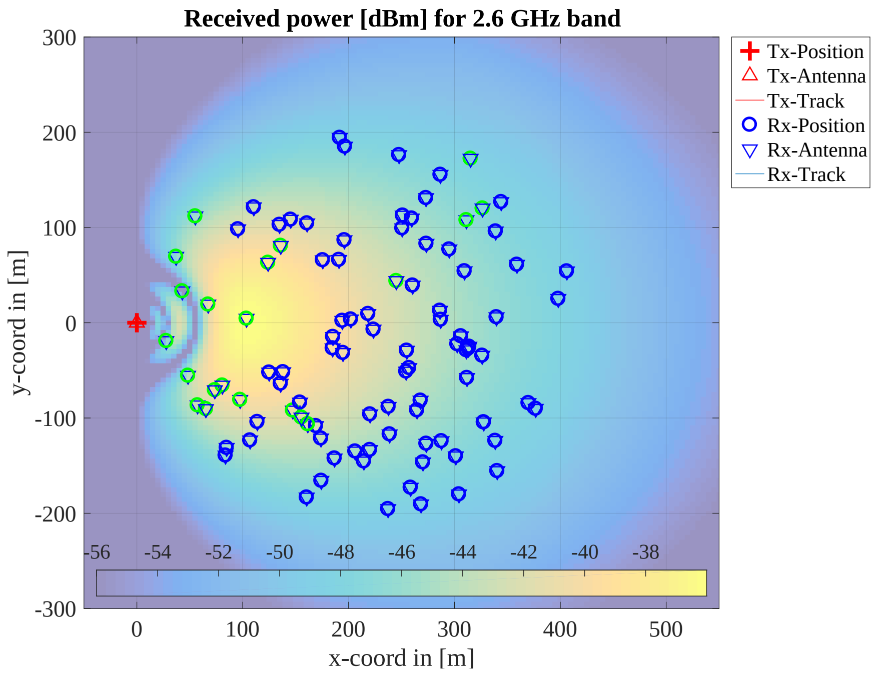
Antenna set-up Two different antenna configurations are used at the BS. The 2.6 GHz antenna is constructed from 8 vertically stacked patch elements with +/- 45 degree polarization. The electric downtilt is set to 8 degree. The mm-wave antenna uses 64 dual-polarized elements in a 8x8 massive-MIMO array configuration. The antennas are assigned to the BS by an array of "qd_arrayant" objects. Rows correspond to the frequency, columns to the BS. There is only 1 BS in the layout. The mobile terminal uses a vertically polarized omni-directional antenna for both frequencies.

```
a_2600_Mhz  = qd_arrayant( '3gpp-3d',  8, 1, s.center_frequency(1), 6, 8 );
a_28000_MHz = qd_arrayant( '3gpp-3d',  8, 8, s.center_frequency(2), 3 );

l.tx_array(1,1) = a_2600_Mhz;                           % Set 2.6 GHz antenna
l.tx_array(2,1) = a_28000_MHz;                          % Set 28 Ghz antenna

l.rx_array = qd_arrayant('omni');                       % Set omni-rx antenna
```

Coverage preview Next, we create a preview of the antenna footprint. We calculate the map for the two frequencies including path-loss and antenna patterns. The first plot is for the 2.6 GHz band.

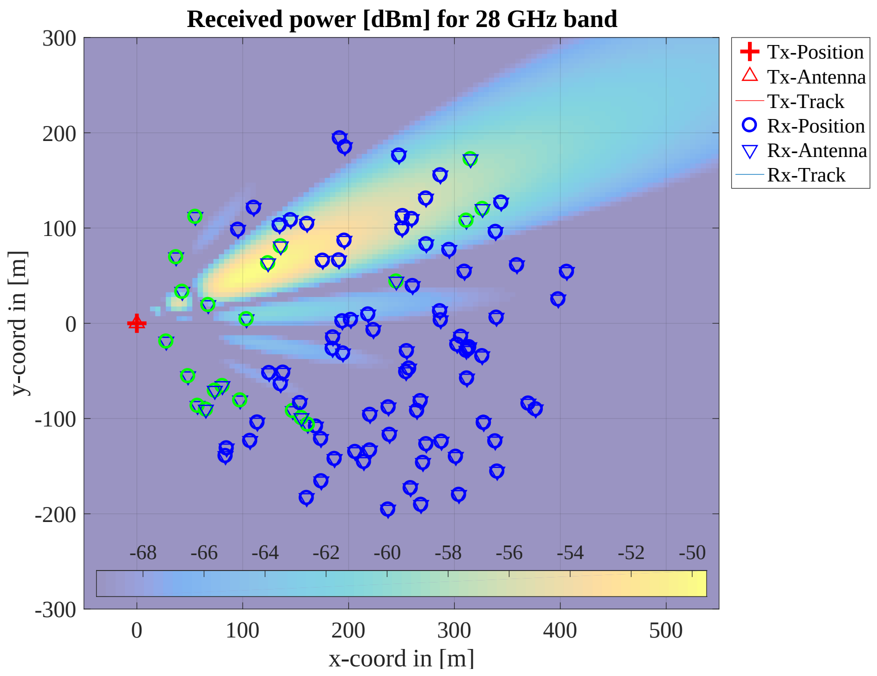
```
sample_distance = 5;                                    % One pixel every 5 m
x_min           = -50;                                  % Area to be samples in [m]
x_max           = 550;
y_min           = -300;
y_max           = 300;
rx_height       = 1.5;                                  % Mobile terminal height in [m]
tx_power        = 30;                                   % Tx-power in [dBm] per antenna element
i_freq          = 1;                                    % Frequency index for 2.6 GHz

% Calculate the map including path-loss and antenna patterns
[ map, x_coords, y_coords] = l.power_map( '3GPP_38.901_UMa_LOS', 'quick',...
    sample_distance, x_min, x_max, y_min, y_max, rx_height, tx_power, i_freq );
P_db = 10*log10( sum( map{1}, 4 ) );

% Plot the results
l.visualize([],[],0);                                   % Show BS and MT positions on the map
hold on; imagesc( x_coords, y_coords, P_db ); hold off  % Plot the antenna footprint
axis([x_min,x_max,y_min,y_max]);
```

```
caxis( max(P_db(:)) + [-20 0] );                     % Color range
colmap = colormap;
colormap( colmap*0.5 + 0.5 );                        % Adjust colors to be "lighter"
set(gca,'layer','top')                               % Show grid on top of the map
colorbar('south')
title('Received power [dBm] for 2.6 GHz band')
```



For the 28 GHz, we get the complex-valued phases for each antenna element in order to calculate a MRT beamformer that points the towards the ground at coordinates x = 200 m and y = 100 m.

```
tx_power        = 10;                                % Tx-power in [dBm] per antenna element
i_freq          = 2;                                 % Frequency index for 28 GHz

% Calculate the map including path-loss and antenna patterns
[ map, x_coords, y_coords] = l.power_map( '3GPP_38.901_UMa_LOS', 'phase',...
    sample_distance, x_min, x_max, y_min, y_max, rx_height, tx_power, i_freq );

% Calculate MRT beamforming weights
beam_x = find( x_coords >= 200 , 1 );                % Point the beam at x = 200 and y = 100
beam_y = find( y_coords >= 100  , 1 );
w = conj( map{1}( beam_y, beam_x , 1 ,: ) );         % Precoding weights for a MRT beamformer
w = w ./ sqrt(mean(abs(w(:)).^2));                   % Normalize to unit power

% Apply the precoding weights to each pixel on the map and calculate the received power
P_db = map{1} .* w( ones(1,numel(y_coords)), ones(1,numel(x_coords)),:,:,: );
P_db = 10*log10( abs( sum( P_db ,4 ) ).^2 );

l.visualize([],[],0);                                % Show BS and MT positions on the map
hold on; imagesc( x_coords, y_coords, P_db ); hold off  % Plot the antenna footprint
axis([x_min,x_max,y_min,y_max]);
caxis( max(P_db(:)) + [-20 0] );                     % Color range
colmap = colormap;
colormap( colmap*0.5 + 0.5 );                        % Adjust colors to be "lighter"
set(gca,'layer','top')                               % Show grid on top of the map
colorbar('south')
title('Received power [dBm] for 28 GHz band')
```



Generate channel coefficients Channel coefficients are generated by calling "l.get_channels". The output is an array of QuaDRiGa channel objects. The first dimension corresponds to the MTs (100). The second dimension corresponds to the number of BSs (1) and the third dimension corresponds to the number of frequencies (2).

```
c = l.get_channels;
```

```
Starting channel generation using QuaDRiGa v2.2.0-0
100 receivers, 1 transmitter, 2 frequencies (2.6 GHz, 28.0 GHz)
Generating channel builder objects - 4 builders, 200 channel segments
Initializing random generators
Generating parameters
SSF Corr.    [oooooooooooooooooooooooooooooooooooooooooooooooooo]      5 seconds
Preparing multi-frequency simulations - 8 builders
Channels     [oooooooooooooooooooooooooooooooooooooooooooooooooo]     12 seconds
Merging      [oooooooooooooooooooooooooooooooooooooooooooooooooo]      0 seconds
Formatting output channels - 200 channel objects
Total runtime: 20 seconds
```

4.12 Ground reflection simulation

This tutorial shows how to include a deterministic ground reflection component into the channel. The effects are then demonstrated for different carrier frequencies (2 GHz, 28 GHz, and 60 GHz).

Simulation assumptions are in accordance with 3GPP 38.901 v14.1.0, Section 7.6.8, p.60 (Explicit ground reflection model). Some modifications are made as described in [Jaeckel, S.; Raschkowski, L.; Wu, S.; Thiele, L. & Keusgen, W.; "An Explicit Ground Reflection Model for mm-Wave Channels", Proc. IEEE WCNC Workshops '17, 2017]. For all ground reflection simulations, a random ground humidity is assumed, which changes the relative permittivity of the ground and, hence, the reflection coefficient will be different for each segment. All ground reflection properties are controlled by the scenario configuration files in the "config" folder of the channel model. The parameter "GR_enabled" activates (1) or deactivates (0) the ground reflection component. The parameter "GR_epsilon" can be used to fix the relative permittivity to a fixed value.

Basic setup Multiple frequencies are set in the simulation parameters by providing a vector of frequency sample points. A new layout is created with a 10 m high BS position. Three different model parametrizations are compared:

- 2-ray ground reflection model without any additional NLOS components
- 3GPP 38.901 Urban Microcell LOS
- Modified 3GPP 38.901 Urban Microcell LOS including a ground reflection

The MT is at 1.5 m height and moves along a 50 m long track starting 10 m away from the BS. The model is set to sample the channel every 10 cm (10 time per meter).

Since the 3GPP scenarios also have non-deterministic NLOS components, there needs to be a birth / death process of the scattering clusters along the MT trajectory. This is done by splitting the track into segments. "split_segment" assumes an average segment length of 30 m with a standard deviation of 5 m.

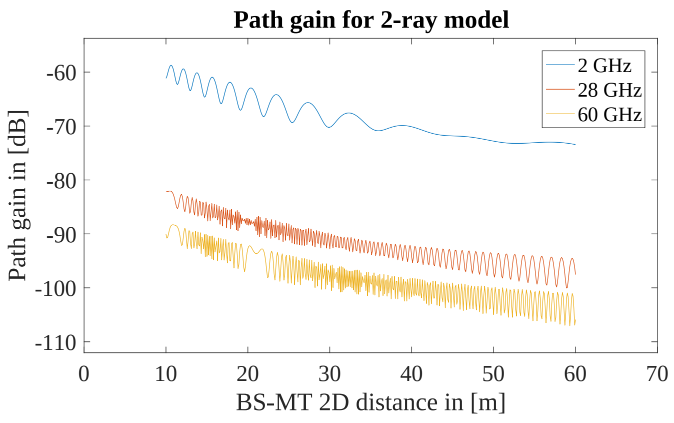
```
close all
clear all

set(0,'defaultTextFontSize', 18)                      % Default Font Size
set(0,'defaultAxesFontSize', 18)                      % Default Font Size
set(0,'defaultAxesFontName','Times')                  % Default Font Type
set(0,'defaultTextFontName','Times')                  % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')        % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')            % Default Paper Type
set(0,'DefaultFigurePaperSize',[14.5 4.5])            % Change paper Size

s = qd_simulation_parameters;
s.center_frequency = [2e9 28e9 60e9];                 % Set the three carrier frequencies

l = qd_layout( s );                                   % New QuaDRiGa layout
l.no_tx = 3;                                          % One BS for each scenario
l.tx_position(3,:) = 10;                              % Set BS height for all scenarios

l.rx_track = qd_track( 'linear' , 50, 0 );            % 50 m long track
l.rx_track.initial_position = [10 ; 0 ; 1.5 ];        % Set start positions and MT height
l.rx_track.interpolate_positions(10);                 % Set sampling rate to 10 saples per meter

% Each of the 3 BS gets assigned a different scenario:
l.rx_track.scenario = { 'TwoRayGR' ; '3GPP_38.901_UMi_LOS' ; '3GPP_38.901_UMi_LOS_GR' };

l.rx_track.split_segment;                             % Split into segments
c = l.get_channels;                                   % Generate the channel coefficients
dist_2d = c(1,1,1).rx_position(1,:);                  % Extract the 2D distance
```

```
Starting channel generation using QuaDRiGa v2.2.0-0
1 receiver, 3 transmitters, 3 frequencies (2.0 GHz, 28.0 GHz, 60.0 GHz)
Warning: Sample density in tracks does not fulfill the sampling theoreme.
Generating channel builder objects - 9 builders, 18 channel segments
Initializing random generators
Generating parameters
SSF Corr.    [oooooooooooooooooooooooooooooooooooooooooooooooooo]     1 seconds
Preparing multi-frequency simulations - 27 builders
Channels     [oooooooooooooooooooooooooooooooooooooooooooooooooo]    19 seconds
Merging      [oooooooooooooooooooooooooooooooooooooooooooooooooo]     2 seconds
Formatting output channels - 9 channel objects
Total runtime: 23 seconds
```

Plot path gain for 2-ray model The first plot shows the results for the 2-ray ground reflection model. One can see the differences in path gain between the 3 frequency bands. The main difference, however, are the rapid power fluctuations due to the interference between the 2 paths. This is very different at mmWave frequencies compared to 2 GHz.

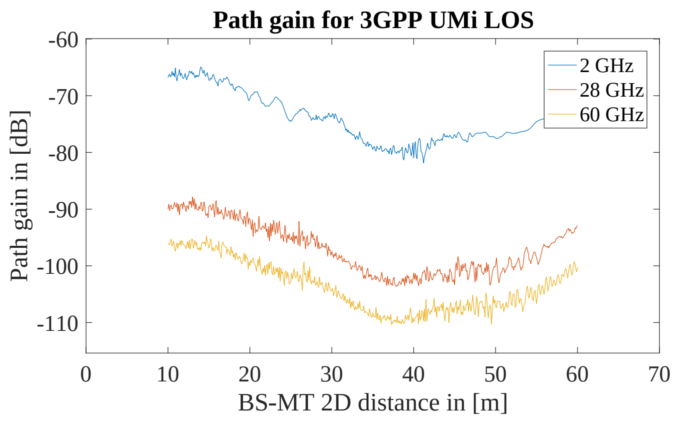
```
H = c(1,1,1).fr(100e6,64);                            % 2 GHz broadband channel (100 MHz)
P_2ray_2Ghz = squeeze(mean(abs(H(1,1,:,:)).^2,3));    % Average power

H = c(1,1,2).fr(100e6,64);                            % 28 GHz broadband channel (100 MHz)
P_2ray_28Ghz = squeeze(mean(abs(H(1,1,:,:)).^2,3));   % Average power
```

```

H = c(1,1,3).fr(100e6,64);                               % 60 GHz broadband channel (100 MHz)
P_2ray_60Ghz = squeeze(mean(abs(H(1,1,:,:)).^2,3));      % Average power

P = 10*log10( [ P_2ray_2Ghz , P_2ray_28Ghz , P_2ray_60Ghz ] );

figure('Position',[ 100 , 100 , 760 , 400]);             % New figure
plot( dist_2d , P )
axis([0, max(dist_2d)+10, min(P(:))-5, max(P(:))+5  ])
xlabel('BS-MT 2D distance in [m]')
ylabel('Path gain in [dB]')
title('Path gain for 2-ray model')
legend('2 GHz','28 GHz','60 GHz')
```



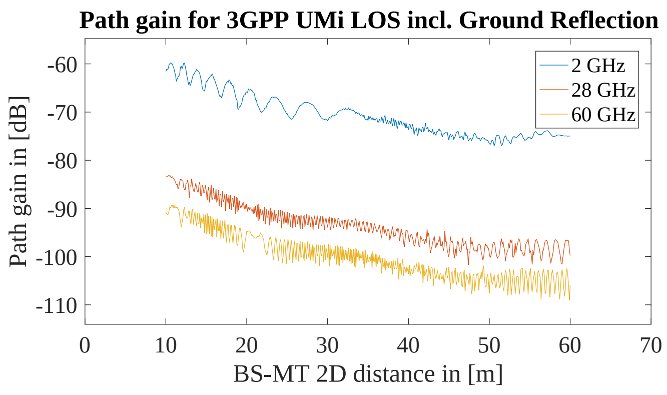
Plot path gain for 3GPP UMi LOS model The second plot sows the results for the 3GPP UMi LOS model. The path loss is similar compared to the 2-ray model. A shadow-fading component induces slow changes in the average received power. By default, the shadow fading is fully correlated between the 3 frequencies. Small-scale-fading correlations are done according to 3GPP TR 38.901 V14.1.0, Section 7.6.5, pp 57. This can be changed by not using "l.get_channels", but executing the channgel generation steps maually in a different order (see the 3GPP TR 38.901 full calibration for more deails). The NLOS components cause some fast fading wihich is averaged out by the broadband processing. No ground reflection is included. Hence, the fast fluctuations are absent.

```
H = c(1,2,1).fr(100e6,64);                               % 2 GHz broadband channel (100 MHz)
P_2ray_2Ghz = squeeze(mean(abs(H(1,1,:,:)).^2,3));       % Average power

H = c(1,2,2).fr(100e6,64);                               % 28 GHz broadband channel (100 MHz)
P_2ray_28Ghz = squeeze(mean(abs(H(1,1,:,:)).^2,3));      % Average power

H = c(1,2,3).fr(100e6,64);                               % 60 GHz broadband channel (100 MHz)
P_2ray_60Ghz = squeeze(mean(abs(H(1,1,:,:)).^2,3));      % Average power

P = 10*log10( [ P_2ray_2Ghz , P_2ray_28Ghz , P_2ray_60Ghz ] );

figure('Position',[ 100 , 100 , 760 , 400]);             % New figure
plot( dist_2d , P )
axis([0, max(dist_2d)+10, min(P(:))-5, max(P(:))+5  ])
xlabel('BS-MT 2D distance in [m]')
ylabel('Path gain in [dB]')
title('Path gain for 3GPP UMi LOS')
legend('2 GHz','28 GHz','60 GHz')
```



Plot path gain for 3GPP UMi LOS model The last plot shows the modified 3GPP channel (see [1]), where the ground reflection is included. Hence, the typical fluctuations are now included.

```
H = c(1,3,1).fr(100e6,64);
P_2ray_2Ghz = squeeze(mean(abs(H(1,1,:,:)).^2,3));

H = c(1,3,2).fr(100e6,64);
P_2ray_28Ghz = squeeze(mean(abs(H(1,1,:,:)).^2,3));

H = c(1,3,3).fr(100e6,64);
P_2ray_60Ghz = squeeze(mean(abs(H(1,1,:,:)).^2,3));

P = 10*log10( [ P_2ray_2Ghz , P_2ray_28Ghz , P_2ray_60Ghz ] );

figure('Position',[ 100 , 100 , 760 , 400]);             % New figure
plot( dist_2d , P )
axis([0, max(dist_2d)+10, min(P(:))-5, max(P(:))+5  ])
xlabel('BS-MT 2D distance in [m]')
ylabel('Path gain in [dB]')
title('Path gain for 3GPP UMi LOS incl. Ground Reflection')
legend('2 GHz','28 GHz','60 GHz')
```



4.13 Spatial consistency

Version 2.0 of the QuaDRiGa channel model supports spatial consistency as specified by 3GPP 38.901 v14.0.0, Section 7.6.3, pp45. This tutorial demonstrates the properties of this feature and how it can be used. Spatial consistency can be seen in three aspects of wireless channels:

- The LOS / NLOS state of a link.
- The large-scale parameters, such as shadow-fading and delay spread
- The positions of the scattering clusters as a function of the mobile terminal (MT) position

Here, points 2 and 3 are covered. The large-scale parameter (point 2) are always spatially consistent. They change slowly when the terminal moves. For example, two MTs that are close together will have similar SFs, DSs and angular spreads. The rate at which the LSPs change is adjusted by the "lambda" parameters in the configuration file. For example: "DS_lambda = 20" means that the delay spread of two terminal at 20 meters distance will be correlated with correlation coefficient of exp(-1) = 0.36. Two terminals at the same positions will see the same DS (correlation coefficient is 1).

The small-scale fading (SSF) is governed by the position of the scattering clusters. Two closely spaced terminals will not only have a similar DS, they will also see the same scattering clusters. This will have an effect on the achievable data rate. QuaDRiGa implements a 3D correlated random process the correlates all random variables that are used to generate the scattering clusters. The decorrelation distance of this process (i.e. the distance where the correlation of the same variable for 2 users drops to 0.36) is controlled by the parameter "SC_lambda" in the configuration files. A value of 0 disables the spatial consistency for SSF.

Model setup and channel generation First, a new layout is created. The center frequency is set to 2 GHz, the BS height is set to 10 m. By default, vertically polarized omni-directional antennas are used.

```
close all
clear all

set(0,'defaultTextFontSize', 18)                          % Default Font Size
set(0,'defaultAxesFontSize', 18)                          % Default Font Size
set(0,'defaultAxesFontName','Times')                      % Default Font Type
set(0,'defaultTextFontName','Times')                      % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')            % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')                % Default Paper Type
set(0,'DefaultFigurePaperSize',[14.5 4.5])                % Change paper Size

l = qd_layout;                                            % Create new QuaDRIGa layout
l.simpar.center_frequency = 2e9;                          % Set center frequency to 2 GHz
l.simpar.use_absolute_delays = 1;                         % Enables true LOS delay
l.simpar.show_progress_bars = 0;                          % Disable progress bars
l.tx_position = [ 0,0,10 ]';                              % Set BS posittions
```

Next, a new receiver trajectory is created. The track is 50 meters long and starts in the north-east of the BS.

```
l.rx_track = qd_track( 'linear' , 50, pi/2 );             % 50 m long track going north
l.rx_track.initial_position = [20 ; 30 ; 1.5 ];           % Set start position and MT height
l.rx_track.interpolate_positions(10);                     % One channel sample every 10 cm
l.rx_track.scenario = '3GPP_38.901_UMi_NLOS';             % Set propagation scenario
```

QuaDRiGa supports two different MT mobility options. By default, drifting is used. This keeps the scattering positions fixed for a short segment of the track. Along a segment, path delays and angles are updated when the terminal is moving. However, 3GPP 38.901 proposed a different mobility option (3GPP 38.901 v14.0.0, Section 7.6.3.2, Option B, pp47). This is implemented in QuaDRiGa as well. It is enabled by setting the number of segments on a track equal to the number of snapshots. Hence, a new channel realization is created for each position on the track. Mobility is then obtained by the spatially consistency procedure.

```
l.rx_track.no_segments = l.rx_track.no_snapshots;         % Use spatial consisteny for mobility
```

Now, a channel builder object is created. The scenario parameters can then be edited to study their effects on the results.

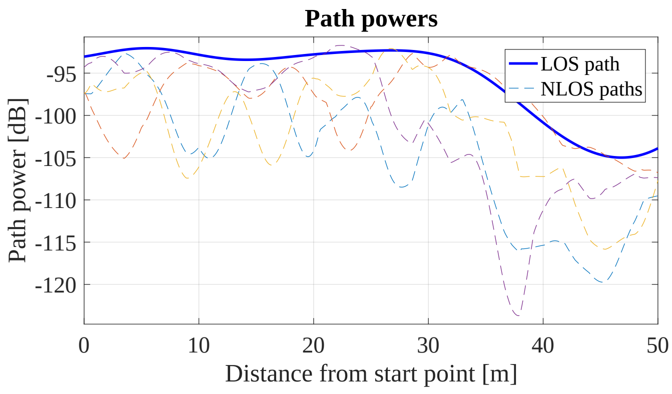
```
b = l.init_builder;                                       % Initializes channel builder
```

3GPP specifies a cluster delay spread for the two strongest clusters (3GPP 38.901 v14.0.0, Table 7.5-5, pp37). When "PerClusterDS" in the configuration file is set to values > 0, the clusters are split into three sub-clusters with different delays. However, this is incompatible with spatial consistency because the strongest cluster changes over time. Therefore, QuaDRiGa applies the cluster delay spread to all clusters which avoids this problem. Here, the cluster delay spread is disabled avoid cluttering the results. You can find out what happens when you set to a different value.

```
b.scenpar.PerClusterDS = 0;                          % Disable per-cluster delay spread
b.scenpar.NumClusters = 5;                           % Only generate 5 clusters
b.scenpar.KF_mu = -3;                                % Set los power to 33 % of the total power
b.scenpar.KF_sigma = 0.5;
b.scenpar.SC_lambda = 5;                             % Set SSF decorrelation distance to 5 m

b.gen_ssf_parameters;                                % Generate small-scale-fading parameters

c = get_channels( b );                               % Generate channel coefficients
c = merge( c, [], 0 );                               % Combine output channels
c.individual_delays = 0;                             % Remove per-antenna delays

dl = c.delay.';                                      % Extract path delays from the channel
pow = squeeze( abs(c.coeff).^2 )';                   % Calculate path powers from the channel

[len,dist] = l.rx_track.get_length;                  % Store the length and distances from start point
```

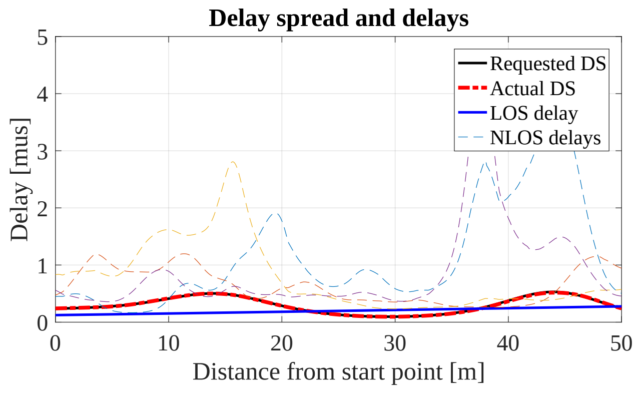
Path powers The first plot shows the path powers along the receiver track. The path parameters (delays, angles, power) are generated as described in 3GPP 38.901 v14.0.0, Section 7.6.3.2, Option B, pp47). As you can see, path powers do not suddenly ”jump”, but they change relatively smoothly when the MT moves.

```
figure('Position',[ 100 , 100 , 760 , 400]);                % New figure
plot( dist,10*log10(pow(:,1)),'-b','Linewidth',2)
hold on; plot( dist,10*log10(pow(:,2:end)),'--'); hold off
axis( [ 0, len, 10*log10(min(pow(:)))-1, 10*log10(max(pow(:)))+1 ] )
grid on
title('Path powers'); xlabel('Distance from start point [m]'); ylabel('Path power [dB]');
legend('LOS path','NLOS paths')
```



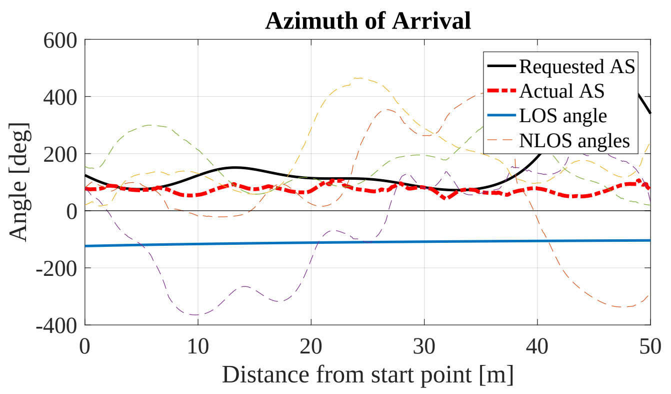
Delay spread The second plot sows the path delays and the delay spread. As for the powers, delays change smoothly over time. NLOS delays can never be smaller than the LOS delay. In addition, the thick black line shows the DS at the input of the model and the red, dashed line shows the DS that is calculated from the channel coefficients. Both should be identical.

```
% Calculate DS from the channel coefficients
pow_normalized = pow ./ (sum(pow,2) * ones( 1,size(pow,2) ));
ds = sqrt( sum( pow_normalized .* dl.^2 , 2 ) - sum( pow_normalized .* dl , 2 ).^2 );

figure('Position',[ 100 , 100 , 760 , 400]);                % New figure
plot( dist,b.ds*1e6,'-k','Linewidth',2 )
hold on
plot( dist,ds*1e6,'-.r','Linewidth',3 )
plot( dist,dl(:,1)*1e6,'-b','Linewidth',2)
plot( dist,dl(:,2:end)*1e6,'--')
hold off; xlim([0,len]); grid on
title('Delay spread and delays'); xlabel('Distance from start point [m]'); ylabel('Delay [mus]')
legend('Requested DS','Actual DS','LOS delay','NLOS delays');
```



Azimuth of Arrival The third plot shows the Azimuth of Arrival (AoA) angles of the paths. As for the DS, the black line shows the angular spread (AS) at the model input and the red dashed line the AS at the output. Those two lines might be different. The arrival angles are distributed on a sphere and therefore, it is not possible to achieve arbitrary angular spreads. At some point the angles just wrap around the circle. Therefore, the maximum AS is limited to vales around 80 degrees.

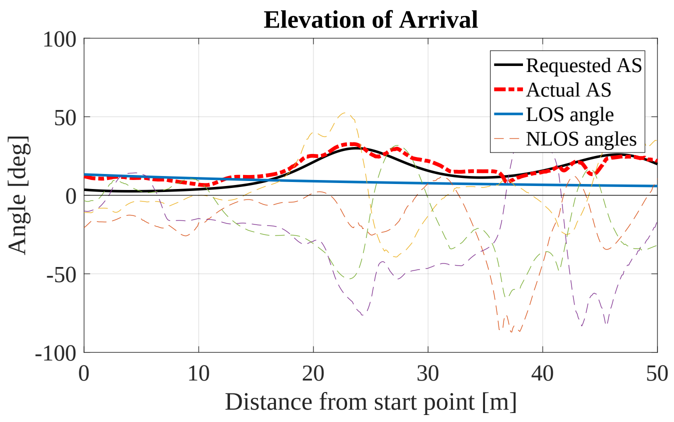
```
% Calculate AS from the channel coefficients
mean_angle = angle( sum( pow_normalized.*exp( 1j*b.AoA ) , 2 ) );
ang = b.AoA - mean_angle * ones( 1,b.NumClusters );
ang = angle( exp( 1j*ang ) );
as = sqrt( sum(pow_normalized.*ang.^2,2) - sum( pow_normalized.*ang,2).^2 ) * 180/pi;

% Unwrap the angles to illustrate spatial consistency
ang_unwrapped = unwrap(b.AoA,1)*180/pi;

figure('Position',[ 100 , 100 , 760 , 400]);              % New figure
plot( dist,b.asA','-k','Linewidth',2 )
hold on
plot( dist,as,'-.r','Linewidth',3 )
plot( dist, ang_unwrapped(:,1) ,'Linewidth',2)
plot( dist, ang_unwrapped(:,2:end),'--')
plot( dist, zeros(b.no_rx_positions,1),'-k')
hold off; xlim([0,len]); grid on
title('Azimuth of Arrival'); xlabel('Distance from start point [m]'); ylabel('Angle [deg]')
legend('Requested AS','Actual AS','LOS angle', 'NLOS angles')
```



Elevation of Arrival Elevation angles are bound between -90 and +90 degrees. The angles also do not change rapidly.

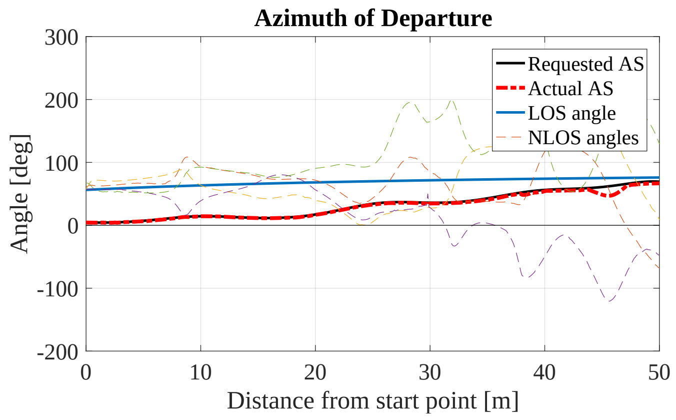
```
% Calculate AS from the channel coefficients
mean_angle = angle( sum( pow_normalized.*exp( 1j*b.EoA ) , 2 ) );
ang = b.EoA - mean_angle * ones( 1,b.NumClusters );
ang = angle( exp( 1j*ang ) );
```

```
as = sqrt( sum(pow_normalized.*ang.^2,2) - sum( pow_normalized.*ang,2).^2 ) * 180/pi;

% Get angles in degres
ang = b.EoA*180/pi;

figure('Position',[ 100 , 100 , 760 , 400]);             % New figure
plot( dist,b.esA','-k','Linewidth',2 )
hold on
plot( dist,as,'-.r','Linewidth',3 )
plot( dist, ang(:,1) ,'Linewidth',2)
plot( dist, ang(:,2:end),'--')
plot( dist, zeros(b.no_rx_positions,1),'-k')
hold off; xlim([0,len]); grid on
title('Elevation of Arrival'); xlabel('Distance from start point [m]'); ylabel('Angle [deg]')
legend('Requested AS','Actual AS','LOS angle', 'NLOS angles')
```



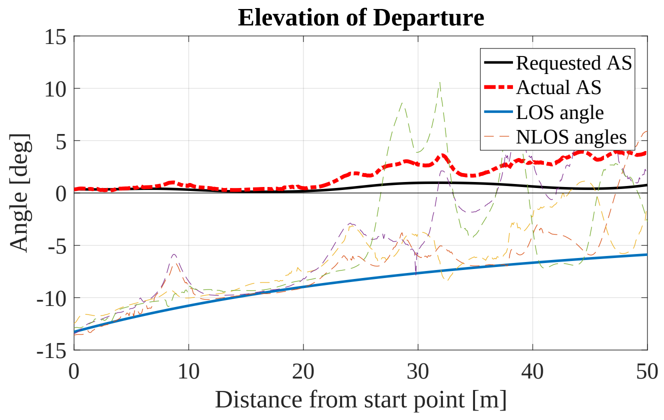
Azimuth of Departure QuaDRiGa calculates the exact positions of the scattering clusters. However, this is not always possible. For example, when the path delay is very short and the departure and arrival angles have too large values, the cluster positions do not exist. In this case, QuaDRiGa uses a single-bounce model, where the departure angles depend on the arrival angles. In this case, the angles of some clusters might suddenly change. However, this happens only for spherical waves. You can deactivate the sperical waves by setting "l.simpar.use_spherical_waves = 0". In this case, no cluster positions are calculated.

```
% Calculate AS from the channel coefficients
mean_angle = angle( sum( pow_normalized.*exp( 1j*b.AoD ) , 2 ) );
ang = b.AoD - mean_angle * ones( 1,b.NumClusters );
ang = angle( exp( 1j*ang ) );
as = sqrt( sum(pow_normalized.*ang.^2,2) - sum( pow_normalized.*ang,2).^2 ) * 180/pi;

% Unwrap the angles to illustrate spatial consistency
ang_unwrapped = unwrap(b.AoD,1)*180/pi;

figure('Position',[ 100 , 100 , 760 , 400]);             % New figure
plot( dist,b.asD','-k','Linewidth',2 )
hold on
plot( dist,as,'-.r','Linewidth',3 )
plot( dist, ang_unwrapped(:,1) ,'Linewidth',2)
plot( dist, ang_unwrapped(:,2:end),'--')
plot( dist, zeros(b.no_rx_positions,1),'-k')
hold off; xlim([0,len]); grid on
title('Azimuth of Departure'); xlabel('Distance from start point [m]'); ylabel('Angle [deg]')
legend('Requested AS','Actual AS','LOS angle', 'NLOS angles')
```



Elevation of Departure Elevation angles are bound between -90 and +90 degrees. The angles also do not change rapidly except for the sudden changes when the model uses single-bounce propagation.

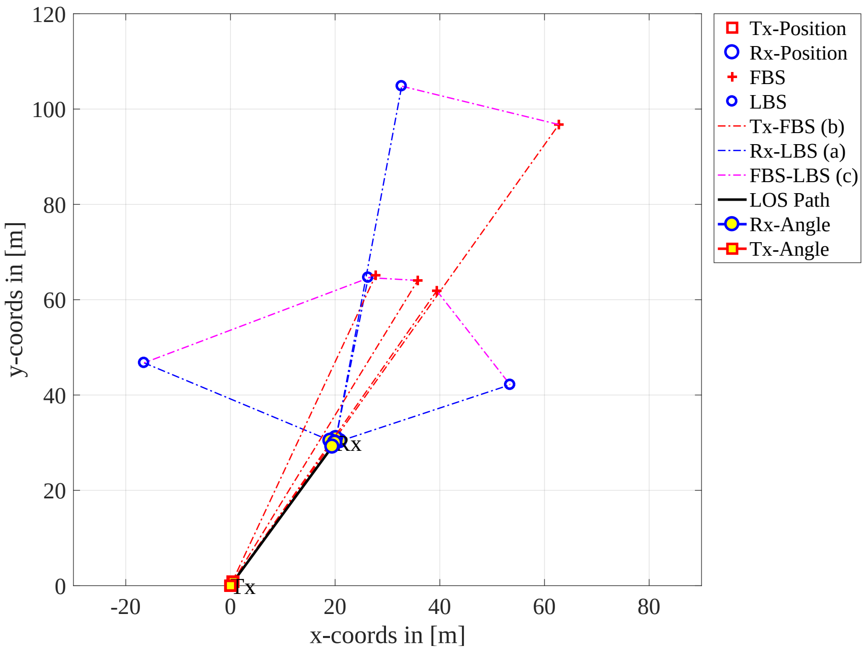
```
% Calculate AS from the channel coefficients
mean_angle = angle( sum( pow_normalized.*exp( 1j*b.EoD ) , 2 ) );
ang = b.EoD - mean_angle * ones( 1,b.NumClusters );
ang = angle( exp( 1j*ang ) );
as = sqrt( sum(pow_normalized.*ang.^2,2) - sum( pow_normalized.*ang,2).^2 ) * 180/pi;

% Get angles in degrees
ang = b.EoD*180/pi;

figure('Position',[ 100 , 100 , 760 , 400]);                % New figure
plot( dist,b.esD','-k','Linewidth',2 )
hold on
plot( dist,as,'-.r','Linewidth',3 )
plot( dist, ang(:,1) ,'Linewidth',2)
plot( dist, ang(:,2:end),'--')
plot( dist, zeros(b.no_rx_positions,1),'-k')
hold off; xlim([0,len]); grid on
title('Elevation of Departure'); xlabel('Distance from start point [m]'); ylabel('Angle [deg]')
legend('Requested AS','Actual AS','LOS angle', 'NLOS angles')
```



Video The last plot shows an visualization of the cluster positions. The spatial consistency model ensures that path delays, angles and power change smoothly with time. However, doe to this, all cluster appear to be moving through the environment. When angles and delays change rapidly, cluster positions change rapidly as well. Sometimes, the speed of the clusters exceed the speed of the MT by several order of magnitude. This violates the WSS conditions which state, that for short time intervals, the cluster positions stay fixed. Hence, a combination of drifting and spatial consistency is needed to achieve realistic channels. (You need to run the code in the loop manually)

```
set(0,'DefaultFigurePaperSize',[14.5 7.3])                 % Default Paper Size
```

```
b.visualize_clusters;
if 0
    for n = 1 : b.no_rx_positions
        b.visualize_clusters(n,[],0);
        title(['Distance from start: ',num2str( dist(n),'%1.1f' ),' m'])
        axis([-100 100 -50 150])
        drawnow
    end
end
```



4.14 Dual Mobility

In order to support moving transmitters and receivers (e.g. for car-to-car or device-to-device communication), QuaDRiGa 2.2 has been extended to support tracks for the transmitter. This tutorial demonstrates how to use the new feature. It covers the following topics:

- Set-up of a layout with two mobile transceivers (cars moving in opposite directions at different speeds) and one fixed BS
- Plot of the coverage area of the fixed BS
- Calculation of the channels for BS-Car1, BS-Car2, and Car1-Car2
- Discussion of computational complexity
- Plot of the path-loss along the trajectory
- Calculation of the Doppler spectrum for the 3 links

An initial set of channel parameters is provided for the Urban-Device-to-Device scenario. Those have been adopted from the Urban-Microcell scenario al low BS heights. However, new measurements are needed for validating the assumptions.

Setting general parameters We set up some basic parameters such as center frequency and sample density. The minimum sample density (samples-per-half-wavelength) must be 1 for static transmitters and 2 for mobile transceivers. This ensures that the Doppler characteristics of the channel can be correctly captured. During the generation of the channel coefficients, interpolation is used to get the correct sample rate (samples-per-second). However, channel interpolation needs much less computing time. Increasing the sample density in the simulation parameters increases the accuracy (less interpolation artefacts) at the cost of much longer simulation times.

```
close all
clear all

set(0,'defaultTextFontSize', 18)                        % Default Font Size
set(0,'defaultAxesFontSize', 18)                        % Default Font Size
set(0,'defaultAxesFontName','Times')                    % Default Font Type
set(0,'defaultTextFontName','Times')                    % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')          % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')              % Default Paper Type
set(0,'DefaultFigurePaperSize',[14.5 7.3])              % Default Paper Size

s = qd_simulation_parameters;                           % New simulation parameters
s.center_frequency = 2.4e9;                             % 2.4 GHz center frequency
s.use_absolute_delays = 1;                              % Include delay of the LOS path
s.sample_density = 2.1;                                 % Minimum possible sample density
```

Defining the layout A new layout is created and the static transmitter is defined first. In QuaDRiGa 2.2, each transmitter has a track. Static transmitters use zero-length tracks. However, it is possible to define a custom orientation for the BS in the track object. Here, the BS is oriented to the north-east.

```
l = qd_layout( s );                                     % New layout

t = qd_track( 'linear', 0 , pi/4 );                     % Static track facing north-east
t.initial_position = [0;0;6];                           % 6 m height
t.name = 'BS';                                          % Assign unique name

a = qd_arrayant( '3gpp-3d', 8, 4, s.center_frequency, 4, 3 );   % High gain antenna
a.coupling = ones(4,1);                                 % Set horizontal coupling
a.combine_pattern;                                      % Combine radiation pattern
a.normalize_gain;                                       % Normalize gain

l.tx_track(1,1) = t;                                    % Assign static tx track
l.tx_array(1,1) = a;                                    % Tx array

% Calculate antenna footprint
[ map, x_coords, y_coords] = l.power_map( '3GPP_38.901_UMi_LOS', 'quick',...
    1, -50, 200, -50, 200, 1.5 );
P_db = 10*log10(map{1});                                % LOS pathloss in dB
```

Next, we create the first mobile transceiver (Car1). It acts as a receiver for the signals from the ”BS” and as a transmitter for ”Car2”. A linear track with 250 m length is created and the speed is set to 100 km/h. Hence, the channel is observed to 9 seconds. For the dual-mobility feature to work, all tracks in the layout must have the same number of snapshots. By default, linear tracks only have a start and an end-point. However, in order to assign segments and scenarios to the track, we need to create intermediate positions. Here, we interpolate the track so that there is a point for each 10 ms, resulting in 901 ”snapshots”. Segments are created along the track using the ”qd_track.set_secenrio” method. The default settings assign a new segment roughly every 30 m. Since ”Car1” is also a transmitter for ”Car2”, the same track is used as a transmitter track. However, segments are only defined for receiver tracks. Transmitter tracks ”inherit” their segmentation from the receiver tracks during the channel generation. For example, if the receiver track for ”Car2” defines a segment from snapshot 200 to snapshot 300, the corresponding snapshots 200 to 300 from the transmitter track are used.

```
t = qd_track( 'linear', 250 , pi/4 );                   % Trajectory of Car 1 moving away from BS
t.set_speed( 100/3.6 );                                 % Speed = 100 km/h
t.interpolate('time',10e-3,[],[],1);                    % Interpolate to to 10 ms grid
t.initial_position = [6;0;1.5];                         % Start position
t.name = 'Car1';                                        % Assign unique name

a = qd_arrayant('dipole');                              % Dipole antenna

l.rx_track(1,1) = t.copy;                               % Assign Rx track 1
l.rx_track(1,1).set_scenario([],[],[]);                 % Create segments (rx-track only)
l.rx_array(1,1) = a;                                    % Assign Rx array 1

l.tx_track(1,2) = t.copy;                               % Assign Rx track 2
```

```
l.tx_array(1,2) = a;                                    % Assign Rx array 2
```

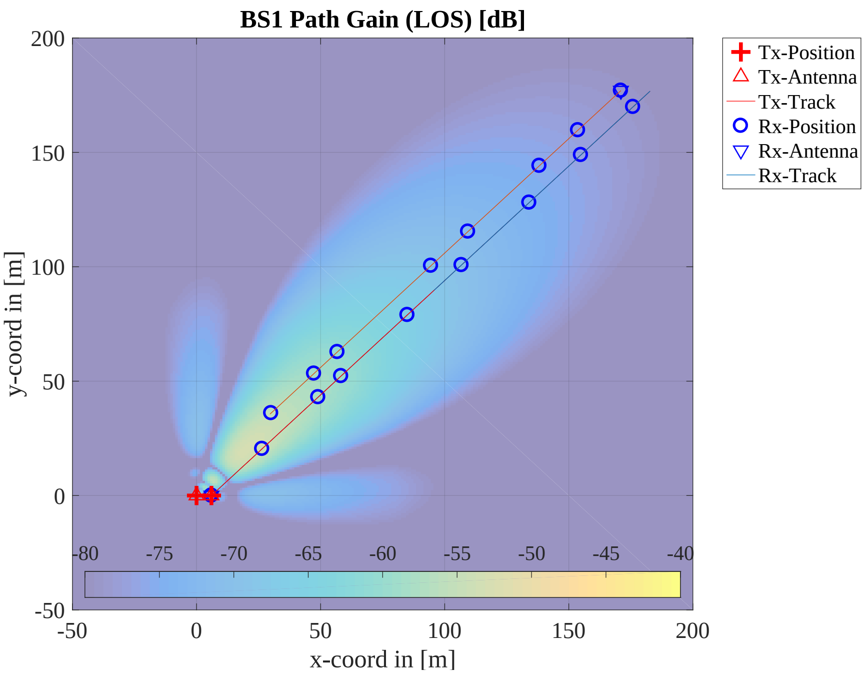
The second mobile receiver "Car2" receives both signals from the "BS" and from "Car1". It travels at 80 km/h in the opposite direction of "Car1". The track length must be shorter due to the lower speed. As for the first track, interpolation is used to obtain 901 snapshots along the track and a different set of segments is created.

```
t = qd_track( 'linear', 200 , -3*pi/4 );                % Trajectory of Car 2 moving towards BS
t.set_speed( 80/3.6 );                                  % Speed = 80 km/h
t.interpolate('time',10e-3,[],[],1);                    % Interpolate to 10 ms grid
t.initial_position = [171;177;1.5];                     % Start position
t.name = 'Car2';                                        % Assign unique name

l.rx_track(1,2) = t;                                    % Assign Rx track 2
l.rx_track(1,2).set_scenario([],[],[]);                 % Create segments (rx-track only)
l.rx_array(1,2) = a;                                    % Assign Rx array 2
```

Now, the scenarios are assigned. The BS-Car links use the default 3GPP Urban-Microcell parameters. For Car-Car channels, we use initial Urban-Device-to-Device parameters. Those have not been confirmed by measurements yet. Since "Car1" acts as both, a transmitter and a receiver, we also need to remove the "Car1-Car1" link from the channel list. Lastly, a plot of the scenario is created showing the BS coverge and the trajectories.

```
l.set_scenario('3GPP_38.901_UMi',[],1,0,40);            % Static transmitter
l.set_scenario('QuaDRiGa_UD2D',  [],2,0,40);            % Mobile tranceivers

l.visualize([],[],0);                                   % Show BS and MT positions on the map
hold on; imagesc( x_coords, y_coords, P_db ); hold off  % Plot the antenna footprint
axis([-50, 200, -50, 200]);
caxis( [-80 -40] );                                     % Color range
colmap = colormap;
colormap( colmap*0.5 + 0.5 );                           % Adjust colors to be "lighter"
set(gca,'layer','top')                                  % Show grid on top of the map
colorbar('south')
title('BS1 Path Gain (LOS) [dB]')
```

```
Setting TX LOS state correlation distance to 40 m
```



Calculate channel coefficients The following command calculates the channel coefficients once per millisecond. The status update is is shown on the command line. This involves the following steps:

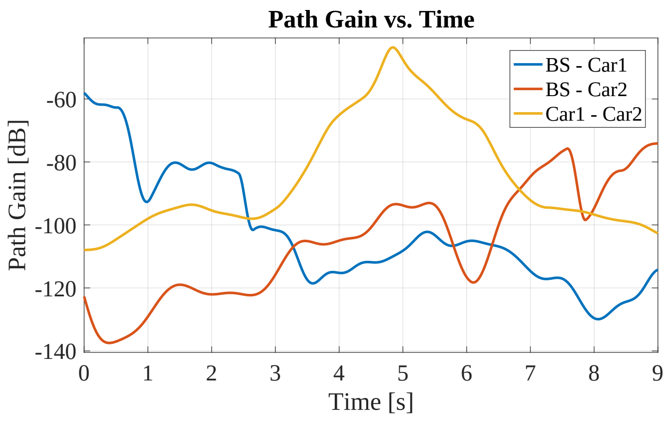
- Interpolation of the tracks to match the sample density. This avoids unnecessary computations but makes sure, that the Doppler profile is completely captured. At 2.4 GHz carrier frequency, 250 m track length, and a sample density of 2.1, 8407 snapshots are needed.
- Generation of channel builder objects and assigning track segments to builders.
- Generation of large and small-scale-fading parameters, including spatial consistency.
- Generation of drifting channel coefficients for each track-segment.
- Merging of channel segments, including modeling the birth and death of scattering clusters.
- Interpolation of channel coefficients to match the sample rate. This generates 9001 snapshots at the output.

```
update_rate = 1e-3;
c = l.get_channels( update_rate );
```

```
Starting channel generation using QuaDRiGa v2.2.0-0
2 receivers, 2 transmitters, 1 frequency (2.4 GHz)
Channel observation time: 9 seconds
Interpolating tracks (v = 27.7778 m/s, SR = 933.9795 samples/s, update factor = 0.934)
Generating channel builder objects - 4 builders, 25 channel segments
Initializing random generators
Generating parameters
SSF Corr.    [oooooooooooooooooooooooooooooooooooooooooooooooo]     2 seconds
Channels     [oooooooooooooooooooooooooooooooooooooooooooooooo]   208 seconds
Merging      [oooooooooooooooooooooooooooooooooooooooooooooooo]    23 seconds
Interpolate  [oooooooooooooooooooooooooooooooooooooooooooooooo]     3 seconds
Formatting output channels - 3 channel objects
Total runtime: 239 seconds
```

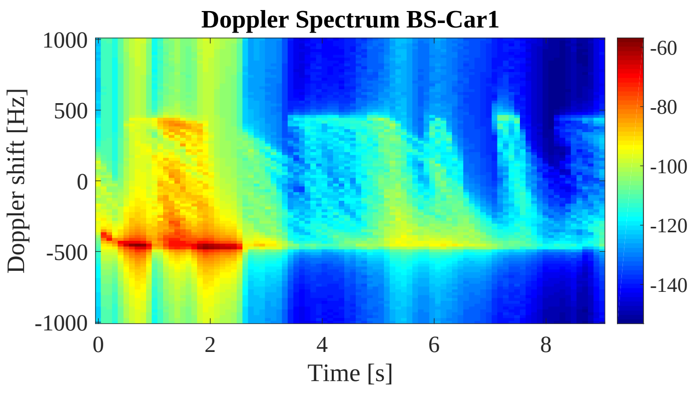
Path gain Now we plot the path-gain for the 3 generated channels. As Car1 moves away from the BS, its PG decreases from roughly -40 dB to about -100 dB. Likewise, the PG of Car2 increases. The PG of the Car1-Car2 channel starts at a low vale and increases until the cars pass each other at about 4.8 seconds simulation time. Then, the PG decreases again.

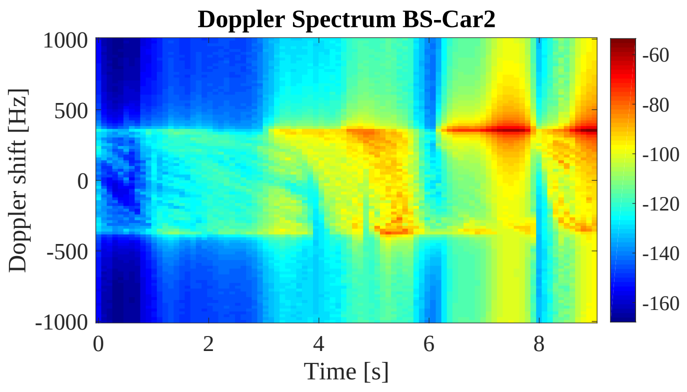
```
time = ( 0 : c(1,1).no_snap-1 ) * update_rate;          % Time axis in seconds
pg = [ c(1,1).par.pg ; c(1,2).par.pg ; c(1,3).par.pg ]; % The path-gain values

set(0,'DefaultFigurePaperSize',[14.5 4.5])              % Change paper Size
figure('Position',[ 100 , 100 , 760 , 400]);            % New figure
plot(time,pg','-','Linewidth',2)                        % Plot target PG
title('Path Gain vs. Time');
xlabel('Time [s]'); ylabel('Path Gain [dB]');
axis([0,max(time),min(pg(:))-3,max(pg(:))+3]); grid on;
legend('BS - Car1','BS - Car2','Car1 - Car2')
```

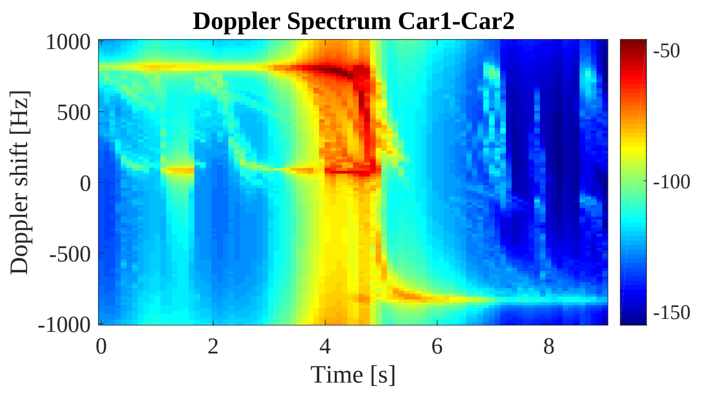


Doppler Spectrum The next three plots show the Doppler spectrum of the three channels. For the BS-Car1 link, the expected Doppler shift (car moving away from BS) is -445 Hz (2*v*fc/c). For BS-Car2, it is 355 Hz and for Car1-Car2 if goes from 800 to -800 Hz when the cars pass each other. Due to the multipath propagation, additional Doppler components occur.

```
w = 100;                                                % Doppler analysis windows size (100 ms)
BW = 100e6;                                             % Channel bandwidth (100 MHz)
N = 128;                                                % Number of carriers

Doppler_axis = -2*( (0:w-1)/(w-1)-0.5)/update_rate;     % The Doppler axis in Hz
time = ( 0 : c(1,1).no_snap-1 ) * update_rate;
Time_axis = time( 1:w:end );                            % Time axis in seconds

no_Doppler = floor( numel(time) ./ w );                 % Number of Doppler samples

for iC = 1 : 3                                          % Repe

    Doppler_spectrum = zeros( w, no_Doppler );          % Preallocate Memory
    for n = 1 : floor( numel(time) ./ w )
        ind = (n-1)*w + 1 : n*w;                        % Snapshot indices
        H = c(1,iC).fr( BW, N, ind );                   % Frequency response of the channel
        H = permute( H,[3,4,1,2] );                     % Reorder dimensions
        G = ifft2(H);                                   % 2D IFFT
        G = fftshift( G,2);                             % Center Doppler spectrum
        Doppler_spectrum( :,n ) = 10*log10( sum( abs(G).^2 , 1 )' );    % Logrithmic power
    end

    figure('Position',[ 100 , 100 , 760 , 400]);        % New figure
    imagesc(Time_axis,Doppler_axis,Doppler_spectrum);   % Create images
    colorbar
    title(['Doppler Spectrum ',regexprep(c(1,iC).name ,'_','-')]);
    xlabel('Time [s]'); ylabel('Doppler shift [Hz]');
    set(gca,'Ydir','Normal')                            % Invert y axis
    colormap jet
end
```






Doppler Spectrum Car1-Car2
1000
500
0
-500
-1000
Doppler shift [Hz]
0
2
4
6
8
Time [s]
-50
-100
-150

5 Model calibration

QuaDRiGa implements all essential components of the 3GPP-3D channel model [9]. In order to qualify as a 3GPP-3D compatible implementation, and thus being eligible to evaluate 3GPP standardization proposals, the model needs to be calibrated. Individual implementations of the 3GPP contributors have to create a set of metrics which show that the model implementation fulfills the 3GPP requirements. This section summarizes the calibration steps and presents the results obtained from QuaDRiGa.

The 3GPP calibration consists of two phases, where the first phase tests the validity of the path-loss model and the elevation angle modeling. The second phase then tests several metrics for the SSF model. The simulation assumptions and results from different 3GPP partners are summarized in [70] and are listed in Table 37.

Table 37: Simulation assumptions for 3GPP-3D calibration

Parameter	Value	
Scenario	3D-UMa	3D-UMi
Layout	Hexagonal grid, 19 micro sites, 3 sectors per site	
ISD	500 m	200 m
BS antenna height	25 m	10 m
Min. BS-MT 2D distance	35 m	10 m
MT indoor fraction	80 %	
MT orientation	Random rotation around z-axis, $r_z \sim \mathcal{U}(0, 360°)$	
MT height in meters	General equation: $h_{MT} = 3(n_{fl} - 1) + 1.5$ Indoor users: $n_{fl} \sim \mathcal{U}(1, N_{fl})$ where $N_{fl} \sim \mathcal{U}(4, 8)$ Outdoor users: $n_{fl} = 1$	
Carrier frequency	2 GHz	
System bandwidth	10 MHz, 50 Resource Blocks	
MT attachment	Strongest BS, based on path loss	
BS antenna (Phase 1)	Config 1: K=M=10, N=1, 0.5λ spacing, V-pol , 12° tilt Config 2: K=M=1, N=1, V-pol	
MT antenna (Phase 1)	Config 1/2: Isotropic antenna, V-pol	
BS antenna (Phase 2)	Config 1: K=1, M=2, N=2, 0.5λ spacing, V-pol Config 2: K=M=10, N=2, X-pol (±45°), 0.5λ spacing, 12° tilt	
MT antenna (Phase 2)	Config 1: N=2, Isotropic antenna, V-pol Config 2: Isotropic antenna, X-pol (0°/90°)	

5.1 3GPP 36.873 Phase 1 Calibration

This section performs the 3GPP calibration as described in 3GPP TR 36.873 V12.5.0, Section 8.2, Page 39 for the phase 1 of the calibration exercise. It is shown how the model is set up to obtain the required results, how the output is processed and how the results compare with the 3GPP baseline. The purpose of the phase 1 calibration is to show the correct working of the path-loss models, the antenna model, the user placement in 3D coordinates.

Antenna setup The antenna model consists of a 2D planar array structure with M rows and N columns of patch elements. Each element has an azimuth and elevation FWHM of 65 degree. The elements can either be vertically polarized or cross-polarizes with plus/minus 45 degree polarization. In the latter, the number of antenna ports is doubled. Optionally, vertically stacked elements can be coupled using fixed complex-valued weights. In order to reduce computational complexity, effective antenna patterns are calculated in QuaDRiGa that include the coupling and downtilt settings.

3GPP uses two antenna configurations for the phase 1 calibration. The first defines a high-gain panel antenna with 10 coupled elements in elevation and 12 degree electric down-tilt. Note: The 102 degree electrical tilt in Table 8.2-1 refer to spheric coordinates, whereas QuaDRiGa uses geographic coordinates. The second antenna is a patch antenna. Both are defined in 3GPP TR 36.873, Section 7.1, Page 17 and implemented "qd_arrayant.generate".

```
clear all
close all

s = qd_simulation_parameters;                   % Set general simulation parameters
s.center_frequency = 2e9;                       % 2 GHz center frequency
s.show_progress_bars = 0;                       % Disable progress bars

% Antenna configuration 1
% 10 elements in elevation, 1 element in azimuth, vertical pol., 12 deg downtilt, 0.5 lambda spacing
a1 = qd_arrayant( '3gpp-3d', 10, 1, s.center_frequency, 4, 12, 0.5 );
a1.element_position(1,:) = 0.5;                 % Distance from pole
a1.name = 'K=M=10';                             % Antenna name

% Antenna configuration 2
% 1 element in elevation, 1 element in azimuth, vertical pol.
a2 = qd_arrayant( '3gpp-3d', 1, 1, s.center_frequency, 1, 0, 0.5 );
a2.element_position(1,:) = 0.5;                 % Distance from pole
a2.name = 'K=M=1';                              % Antenna name
```

QuaDRiGa Setup Here, the channel model is configured. The simulation assumptions are given in Table 8.2-1 in 3GPP TR 36.873 V12.5.0. 3GPP specifies to perform simulations for 3D-UMa and 3D-UMi. The scenario parameters are given in Table 6.1, page 14. Combined with the two antenna configurations, there are four simulation setups. Hence, we define 4 QuaDRIGa layouts. All 3GPP scenarios define a a hexagonal grid with 19 sites and three sectors per site. This is implemented in "qd_layout.generate", using the "regular" layout.

```
no_rx = 2000;                                   % Number of MTs (directly scales the simulation time)
create_curves = 1:4;                            % The number of curves to create

s.use_3GPP_baseline = 1;                        % Disable spherical waves and geometric polarization

isd = [ 200, 200, 500, 500 ];                                  % ISD in each layout
no_go_dist = [ 10, 10, 35, 35 ];                               % Min. UE-eNB 2D distance

l(1,1) = qd_layout.generate( 'regular', 19, isd(1), a2);      % 200 m ISD, K=M=1
l(1,1).simpar = s;                                             % Set simulation parameters
l(1,1).tx_position(3,:) = 10;                                  % 10 m BS height
l(1,1).name = '3D-UMi (K=M=1)';

l(1,2) = qd_layout.generate( 'regular', 19, isd(2), a1);      % 200 m ISD, K=M=10
l(1,2).tx_position(3,:) = 10;                                  % 10 m BS height
l(1,2).simpar = s;                                             % Set simulation parameters
l(1,2).name = '3D-UMi (K=M=10)';

l(1,3) = qd_layout.generate( 'regular', 19, isd(3), a2);      % 500 m ISD, K=M=1
l(1,3).tx_position(3,:) = 25;                                  % 25 m BS height
l(1,3).simpar = s;                                             % Set simulation parameters
l(1,3).name = '3D-UMa (K=M=1)';

l(1,4) = qd_layout.generate( 'regular', 19, isd(4), a1);      % 500 m ISD, K=M=10
l(1,4).tx_position(3,:) = 25;                                  % 25 m BS height
l(1,4).simpar = s;                                             % Set simulation parameters
l(1,4).name = '3D-UMa (K=M=10)';

% Drop users in each layout
for il = create_curves
    l(1,il).no_rx = no_rx;                                     % Number of users
    l(1,il).randomize_rx_positions( 0.93*isd(il), 1.5, 1.5, 0 );  % Random positions in first ring

    % Keep no-go distance
    ind = find(abs( l(1,il).rx_position(1,:) + 1j*l(1,il).rx_position(2,:) ) < no_go_dist(il));
    while ~isempty( ind )
        l(1,il).randomize_rx_positions( 0.93*isd(il), 1.5, 1.5, 0, ind );
        ind = find( abs(l(1,il).rx_position(1,:) + 1j*l(1,il).rx_position(2,:) ) < no_go_dist(il));
    end

    % Set random height of the users
    floor = randi(5,1,l(1,il).no_rx) + 3;                      % Number of floors in the building
    for n = 1 : l(1,il).no_rx
        floor( n ) =  randi(  floor( n ) );                    % Floor level of the UE
    end
    l(1,il).rx_position(3,:) = 3*(floor-1) + 1.5;              % Height in meters

    % Set the scenario and assign LOS probabilities (80% of the users are inddor)
    % "set_scenario" returns an indicator if the user is indoors (1) or outdoors (0)
    switch il
        case {1,2} % UMi
            indoor_rx = l(1,il).set_scenario('3GPP_3D_UMi',[],[],0.8);
        case {3,4} % UMa
            indoor_rx = l(1,il).set_scenario('3GPP_3D_UMa',[],[],0.8);
    end
    l(1,il).rx_position(3,~indoor_rx) = 1.5;                   % Set outdoor-users to 1.5 m height

    % Set user antenna
    l(1,il).rx_array = qd_arrayant('omni');                    % Omni-Antenna, vertically polarized
end
```

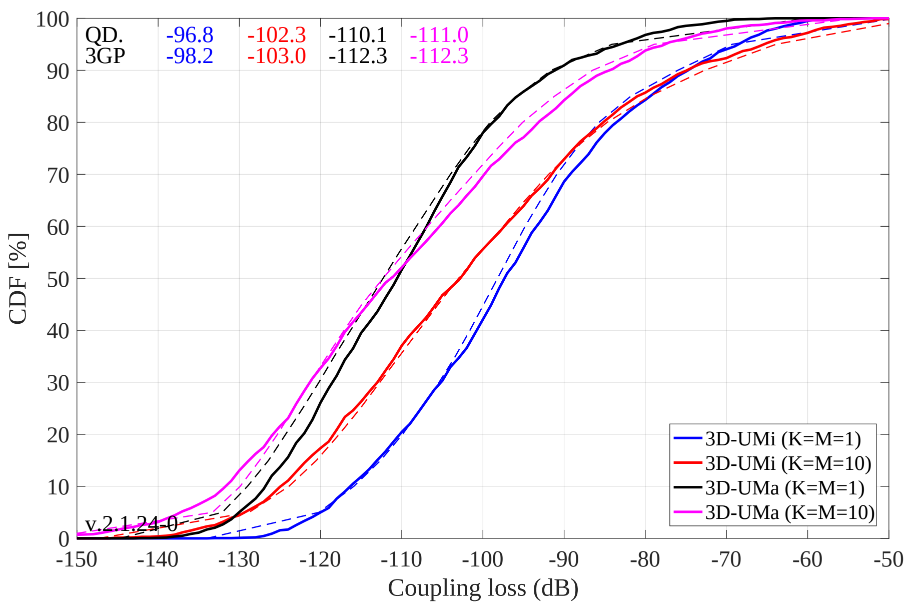
Generate channels Now, the required metric are generated by the model. The MT is always connected to the strongest serving BS. The coupling loss describes the received power to this BS relative to 0 dBm transmit power. Only the LOS path is considered. Other metrics are the geometry factor (GF) and the zenith angle at the BS.

```
tic
pg_eff =zeros( no_rx, 19*3, 4 );                               % Effective PG for each MT and BS
zod    = zeros( no_rx*19, 4 );                                 % Zenith angles for each MT and BS site
for il = create_curves
    coeff  = zeros( no_rx * 19 , 3 );                          % Raw channel coefficients
    name   = cell( no_rx * 19, 1  );                           % Name in the form "Tx_Rx"

    b = l(1,il).init_builder;                                  % Initialze channel builder objects
    gen_lsf_parameters( b );                                   % Generat shadow fading
    cf = get_los_channels( b );                                % Get the LOS channel coefficients only

    cnt = 1;                                                   % Counter
    sic = size(b);
    for i_cb = 1 : numel(b)
        [ i1,i2 ] = qf.qind2sub( sic, i_cb );
        tx_name = ['Tx',num2str(i2,'%02d')];                   % Tx name, e.g. "Tx01"

        if b(i1,i2).no_rx_positions > 1
            tmp = b(i1,i2).get_angles;                         % 3D angles btween BS and MT
            zod( cnt : cnt+b(i1,i2).no_rx_positions-1,il) = 90-tmp(3,:);
        end

        for i_mt = 1 : b(i1,i2).no_rx_positions
            rx_name = b( i1,i2 ).rx_track(1,i_mt).name;        % Rx name, e.g. "Rx0001"
            name{ cnt }  = [tx_name,'_',rx_name];              % Link name, e.g. "Tx01_Rx0001"
            coeff(cnt,:) = cf(i1,i2).coeff(1,:,1,i_mt);        % Channel coefficients
            cnt = cnt + 1;                                     % Increase counter
        end
    end

    [~,ii] = sort( name );                                     % Get the correct order of the channels
    zod(:,il) = zod(ii,il);                                    % Sort ZODs by name

    tmp = reshape( coeff(ii,:), no_rx, 19, 3 );                % Split the 3 sectors from each BS site
    tmp = permute( tmp, [1,3,2] );                             % Reorder the channels
    pg_eff(:,:,il) = reshape( tmp, no_rx, [] );
end
pg_eff = abs( pg_eff ).^2;                                     % Amplitude --> Power
zod = reshape( zod, no_rx, 19, 4 );
toc
```

```
Elapsed time is 106.073939 seconds.
```

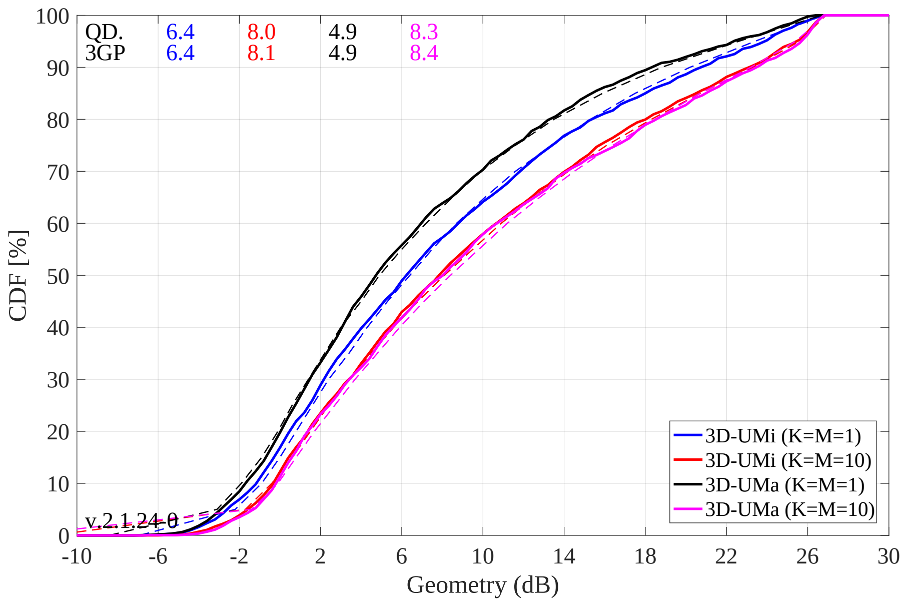
Coupling Loss The coupling loss is defined as the path gain of a MT to its serving BS, i.e. the strongest BS seen by the MT. Here, the term BS refers to one sector of a 3-sector site. In the proposed layout, there are 19 sites, each consisting of three BSs. MTs were placed in the first ring of interferers, i.e. around the first site. The phase 1 calibration does not consider a SSF model, but includes the antenna patterns. Hence, the results shown in the following figure were obtained by running the simulations with only one path (the LOS path). The thick lines were obtained using the QuaDRiGa model, the thin dashed line are taken from 3GPP 36.873. They represent the median of all 3GPP calibration results. The QuaDRiGa results fit almost perfectly. The remaining differences are well within the tolerances visible in the individual result curves.

```
calib_3GPP_ref_data;                                                    % Load reference data

legend_names = { l(1,1).name, l(1,2).name, l(1,3).name, l(1,4).name };  % Legend entries
line_col = {'b','r','k','m'};                                           % Color of the lines

set(0,'defaultTextFontSize', 18)                                        % Default Font Size
set(0,'defaultAxesFontSize', 18)                                        % Default Font Size
set(0,'defaultAxesFontName','Times')                                    % Default Font Type
set(0,'defaultTextFontName','Times')                                    % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')                          % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')                              % Default Paper Type
set(0,'DefaultFigurePaperSize',[14.5 6.6])                              % Default Paper Size

% Calculate the coupling loss from the effective PG
coupling_loss = zeros( no_rx, 4 );
for il = create_curves
    coupling_loss(:,il) = 10*log10(max( pg_eff(:,:,il),[],2 ));
end

figure('Position',[ 50 , 550 , 950 , 600]);
axes('position',[0.09 0.12 0.88 0.86]); hold on;

xm = -150; wx = 100; tx = 0.01; ty = 97;
text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
ln = []; bins = (-0.1:0.01:1.1)*wx+xm;
for il = create_curves
    ln(end+1) = plot( bins, 100*qf.acdf(coupling_loss(:,il),bins),['-',line_col{il}],'Linewidth',2);
    plot( cl36873a(il,:), 0:5:100,['--',line_col{il}],'Linewidth',1 )
    text((tx+0.1*il)*wx+xm,ty,num2str(median(coupling_loss(:,il)),'%1.1f'),'Color',line_col{il});
    text((tx+0.1*il)*wx+xm,ty-4,num2str(cl36873a(il,11),'%1.1f'),'Color',line_col{il});
end

hold off; grid on; box on;
set(gca,'YTick',0:10:100); set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
xlabel('Coupling loss (dB)')
ylabel('CDF [%]')
legend(ln,legend_names( create_curves ),'Location', 'SouthEast')
text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
```



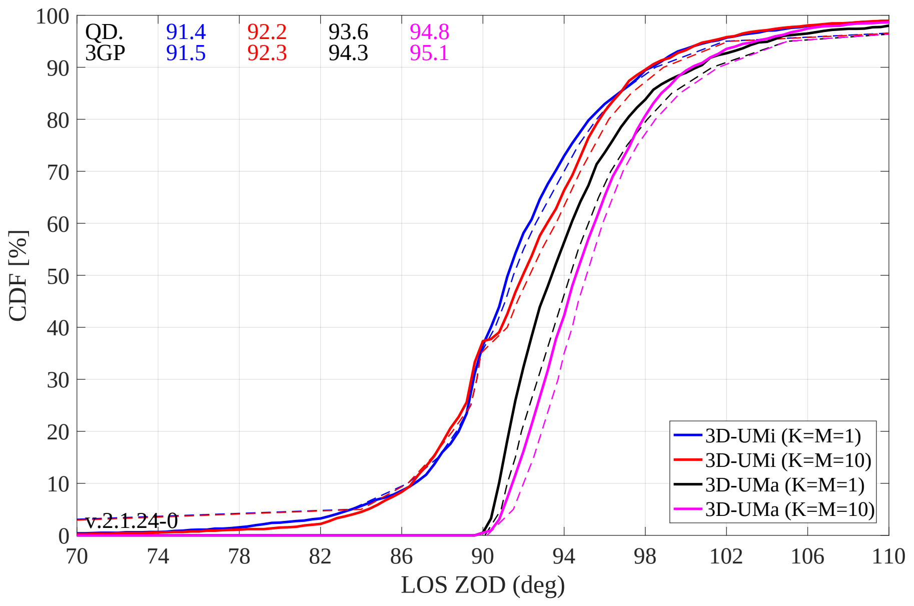
Geometry Factor The GF is a lower bound for the actual SINR. It is defined as the power ratio of the serving BS and the sum power of all interfering BSs. The results in the following Figure agree well with the 3GPP calibrations curves.

```
% Calculate the GF
gf = zeros( no_rx, 4 );
for il = create_curves
    gf(:,il) = 10*log10( max( pg_eff(:,:,il),[],2 ) ./ ( sum( pg_eff(:,:,il),2 ) -...
        max( pg_eff(:,:,il),[],2 ) ) );
end

figure('Position',[ 50 , 550 , 950 , 600]);
axes('position',[0.09 0.12 0.88 0.86]); hold on;

xm = -10; wx = 40; tx = 0.01; ty = 97;
text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
ln = []; bins = (-0.1:0.01:1.1)*wx+xm;
for il = create_curves
    ln(end+1) = plot( bins, 100*qf.acdf(gf(:,il),bins),['-',line_col{il}],'Linewidth',2);
    plot( gf36873a(il,:), 0:5:100,['--',line_col{il}],'Linewidth',1 )
    text((tx+0.1*il)*wx+xm,ty,num2str(median(gf(:,il)),'%1.1f'),'Color',line_col{il});
    text((tx+0.1*il)*wx+xm,ty-4,num2str(gf36873a(il,11),'%1.1f'),'Color',line_col{il});
end

hold off; grid on; box on;
set(gca,'YTick',0:10:100); set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
xlabel('Geometry (dB)')
ylabel('CDF [%]')
legend(ln,legend_names( create_curves ),'Location', 'SouthEast')
text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
```



Evaluation: Zenith of Departure Angle The ZoD is calculated from the LOS path between the serving BS and the MT position. The values in the follwing Figure prove that the model is a 3D model. Users are placed on different floors and the serving BS is determined based on the power of the LOS path. Note that this power value changes when different antenna patterns are used. Hence, the assignment of MTs to BSs is different, depending on which antennas are used at the BS, which explains why the curves differ from each other. The results obtained from QuaDRiGa agree almost perfectly with the 3GPP calibration curves (tolerances are within 0.1 degree).

```
% Determine the serving site
zod_serving = zeros( no_rx , 4 );
for il = create_curves
    [~,serving] = max(pg_eff(:,:,il),[],2);
    serving = ceil( serving / 3 - 0.1 );
    for ir = 1 : no_rx
        zod_serving( ir,il ) = zod( ir, serving( ir ),il );
    end
end

figure('Position',[ 50 , 550 , 950 , 600]);
axes('position',[0.09 0.12 0.88 0.86]); hold on;

xm = 70; wx = 40; tx = 0.01; ty = 97;
text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
ln = []; bins = (-0.1:0.01:1.1)*wx+xm;
for il = create_curves
    ln(end+1) = plot( bins, 100*qf.acdf(zod_serving(:,il),bins),['-',line_col{il}],'Linewidth',2);
    plot( zod36873a(il,:), 0:5:100,['--',line_col{il}],'Linewidth',1 )
    text((tx+0.1*il)*wx+xm,ty,num2str(median(zod_serving(:,il)),'%1.1f'),'Color',line_col{il});
    text((tx+0.1*il)*wx+xm,ty-4,num2str(zod36873a(il,11),'%1.1f'),'Color',line_col{il});
end

hold off; grid on; box on;
set(gca,'YTick',0:10:100); set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
xlabel('LOS ZOD (deg)')
ylabel('CDF [%]')
legend(ln,legend_names( create_curves ),'Location', 'SouthEast')
text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
```



5.2 3GPP 36.873 Phase 2 Calibration

This section performs the 3GPP calibration as described in 3GPP TR 36.873 V12.5.0, Section 8.2, Page 42 for the phase 2 of the calibration exercise. It is shown how the model is set up to obtain the required results, how the output is processed and how the results compare with the 3GPP baseline.

Antenna setup 3GPP uses two antenna configurations for the phase 2 calibration. The first BS array antenna is an 2x2 array of vertically polarized patch antennas (0.5 lambda spacing). The second antenna is a high-gain panel antenna with 10 coupled elements in elevation and two plus/minus 45 degree polarized columns. The electric downtilt is set to 12 degree. Note: The 102 degree electrical tilt in Table 8.2-2 refer to spheric coordinates, whereas QuaDRiGa uses geographic coordinates. The first MS antenna is an two-element ULA with vertical polarization. The second antenna is an 0/90 degree cross-polarized array antenna.

```
clear all
close all

s = qd_simulation_parameters;                 % Set general simulation parameters
s.center_frequency = 2e9;                     % 2 GHz center frequency

% BS antenna configuration 1
% 2 elements in elevation, 2 elements in azimuth, vertical pol., 0.5 lambda spacing
a_bs_1 = qd_arrayant( '3gpp-3d', 2, 2, s.center_frequency, 1, 0, 0.5 );
a_bs_1.element_position(1,:) = 0.5;           % Distance from pole
a_bs_1.name = 'K=1, M=2';                     % Antenna name

% BS antenna configuration 2
% 10 elements in elevation, 2 element in azimuth, vertical pol., 12 deg downtilt, 0.5 lambda spacing
a_bs_2 = qd_arrayant( '3gpp-3d', 10, 2, s.center_frequency, 6, 12, 0.5 );
a_bs_2.element_position(1,:) = 0.5;          % Distance from pole
a_bs_2.name = 'K=M=10';                       % Antenna name

% MT antenna configuration 1
% 1 element in elevation, 2 elements in azimuth, vertical pol., 0.5 lambda spacing
a_mt_1 = qd_arrayant('omni');
a_mt_1.copy_element(1,2);
a_mt_1.element_position(2,:) = [ -s.wavelength/2 , s.wavelength/2 ]*0.5;

% MT antenna configuration 2
% 1 element in elevation, 2 elements in azimuth, X-pol. 0/90, 0.5 lambda spacing
a_mt_2 = qd_arrayant('omni');
```

```
a_mt_2.copy_element(1,2);
a_mt_2.Fa(:,:,2) = 0;
a_mt_2.Fb(:,:,2) = 1;
```

QuaDRiGa Setup Here, the channel model is configured. The simulation assumptions are given in Table 8.2-2 in 3GPP TR 36.873 V12.5.0. 3GPP specifies to perform simulations for 3D-UMa and 3D-UMi. The scenario parameters are given in Table 6.1, page 14. Combined with the two antenna configurations, there are four simulation setups. Hence, we define four QuaDRIGa layouts. All 3GPP scenarios define a a hexagonal grid with 19 sites and three sectors per site. This is implemented in "qd_layout.generate", using the "regular" layout.

```
no_rx = 2000;                                   % Number of MTs (directly scales the simulation time)
create_curves = 1:4;                            % The number of curves to create

s.use_3GPP_baseline = 1;                        % Disable spherical waves
s.show_progress_bars = 0;                       % Enable / disable status display

isd = [ 200, 200, 500, 500 ];                                       % ISD in each layout
no_go_dist = [ 10, 10, 35, 35 ];                                    % Min. UE-eNB 2D distance

l(1,1) = qd_layout.generate( 'regular', 19, isd(1), a_bs_1);      % 200 m ISD, K=M=1
l(1,1).simpar = s;                                                  % Set simulation parameters
l(1,1).tx_position(3,:) = 10;                                       % 10 m BS height
l(1,1).name = '3D-UMi (K=1,M=2)';

l(1,2) = qd_layout.generate( 'regular', 19, isd(2), a_bs_2);      % 200 m ISD, K=M=10
l(1,2).tx_position(3,:) = 10;                                       % 10 m BS height
l(1,2).simpar = s;                                                  % Set simulation parameters
l(1,2).name = '3D-UMi (K=M=10)';

l(1,3) = qd_layout.generate( 'regular', 19, isd(3), a_bs_1);      % 500 m ISD, K=M=1
l(1,3).tx_position(3,:) = 25;                                       % 25 m BS height
l(1,3).simpar = s;                                                  % Set simulation parameters
l(1,3).name = '3D-UMa (K=1,M=2)';

l(1,4) = qd_layout.generate( 'regular', 19, isd(4), a_bs_2);      % 500 m ISD, K=M=10
l(1,4).tx_position(3,:) = 25;                                       % 25 m BS height
l(1,4).simpar = s;                                                  % Set simulation parameters
l(1,4).name = '3D-UMa (K=M=10)';

% Dorp users in each layout
for il = create_curves
    l(1,il).no_rx = no_rx;                                          % Number of users
    l(1,il).randomize_rx_positions( 0.93*isd(il),1.5,1.5,0 );      % Random positions in first ring

    % Keep no-go distance
    ind = find(abs( l(1,il).rx_position(1,:) + 1j*l(1,il).rx_position(2,:) ) < no_go_dist(il));
    while ~isempty( ind )
        l(1,il).randomize_rx_positions( 0.93*isd(il), 1.5, 1.5, 0, ind );
        ind = find( abs(l(1,il).rx_position(1,:) + 1j*l(1,il).rx_position(2,:) ) < no_go_dist(il));
    end

    % Set random height of the users
    floor = randi(5,1,l(1,il).no_rx) + 3;                           % Number of floors in the building
    for n = 1 : l(1,il).no_rx
        floor( n ) =  randi(  floor( n ) );                         % Floor level of the UE
    end
    l(1,il).rx_position(3,:) = 3*(floor-1) + 1.5;                   % Height in meters

    % Set the scenario and assign LOS probabilities (80% of the users are inddor)
    % "set_scenario" returns an indicator if the user is indoors (1) or outdoors (0)
    switch il
        case {1,2} % UMi
            indoor_rx = l(1,il).set_scenario('3GPP_3D_UMi',[],[],0.8);

        case {3,4} % UMa
            indoor_rx = l(1,il).set_scenario('3GPP_3D_UMa',[],[],0.8);
    end
    l(1,il).rx_position(3,~indoor_rx) = 1.5;                        % Set outdoor-users to 1.5 m height

    switch il                                                       % Set user antenna
```

```
        case {1,3} % ULA
            l(1,il).rx_array = a_mt_1;
        case {2,4} % X-POL
            l(1,il).rx_array = a_mt_2;
    end
end
```

Generate channels Channels are now generated using the default QuaDRiGa method (phase 1 only used the LOS path). This will take quite some time.

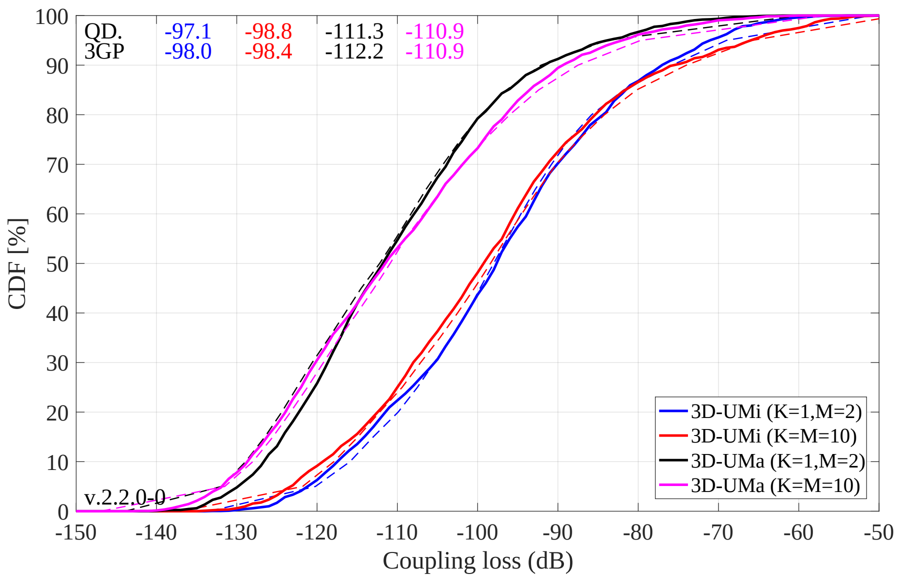
```
tic                                                     % Time the simulations
clear c
for il = create_curves
    cl = l(1,il).get_channels;                          % Generate channels
    nEl = l(1,il).tx_array(1,1).no_elements / 3;        % Number of elements per sector
    nEl = { 1:nEl , nEl+1:2*nEl , 2*nEl+1:3*nEl  };     % Element indices per sector
    c(:,:,il) = split_tx( cl,nEl );                     % Split channels from each sector
end
toc
```

```
Elapsed time is 2145.225497 seconds.
```

Coupling Loss In the second phase of the calibration, the SSF model is enabled. Hence, all NLOS paths are included in the evaluations. For this reason, the coupling loss changes compared to phase 1. Multiple paths are now differently weighted by the antenna pattern, depending on the departure angles at the BS. The path gain is calculated by averaging the power of all sublinks of the MIMO channel matrix. As for phase 1, the coupling loss is the path gain of the serving BS. MTs are assigned to BSs based on the maximum path gain value.

```
calib_3GPP_ref_data;                                                    % Load reference data

legend_names = { l(1,1).name, l(1,2).name, l(1,3).name, l(1,4).name };  % Legend entries
line_col = {'b','r','k','m'};                                           % Color of the lines

set(0,'defaultTextFontSize', 18)                                        % Default Font Size
set(0,'defaultAxesFontSize', 18)                                        % Default Font Size
set(0,'defaultAxesFontName','Times')                                    % Default Font Type
set(0,'defaultTextFontName','Times')                                    % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')                          % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')                              % Default Paper Type
set(0,'DefaultFigurePaperSize',[14.5 6.6])                              % Default Paper Size

pg_eff = zeros( no_rx, 19*3, 4 );       % Calculate the effective path gain from the channels
for il = create_curves
    % Get the number of MIMO sob-channels in the channel matrix
    no_mimo_links = l(1,il).tx_array(1,1).no_elements / 3 * l(1,il).rx_array(1,1).no_elements;
    for ir = 1 : no_rx                                  % Extract effective PG vor each BS-MT link
        for it = 1 : 19*3
            pg_eff( ir,it,il ) = sum( abs(c(ir,it,il).coeff(:)).^2 )/no_mimo_links;
        end
    end
end

coupling_loss = zeros( no_rx, 4 );      % Calculate the coupling loss from the effective PG
for il = create_curves
    coupling_loss(:,il) = 10*log10(max( pg_eff(:,:,il),[],2 ));
end

figure('Position',[ 50 , 550 , 950 , 600]);
axes('position',[0.09 0.12 0.88 0.86]); hold on;

xm = -150; wx = 100; tx = 0.01; ty = 97;
text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
ln = []; bins = (-0.1:0.01:1.1)*wx+xm;
for il = create_curves
    ln(end+1) = plot( bins, 100*qf.acdf(coupling_loss(:,il),bins),['-',line_col{il}],'Linewidth',2);
    plot( cl36873b(il,:), 0:5:100,['--',line_col{il}],'Linewidth',1 )
    text((tx+0.1*il)*wx+xm,ty,num2str(median(coupling_loss(:,il)),'%1.1f'),'Color',line_col{il});
    text((tx+0.1*il)*wx+xm,ty-4,num2str(cl36873b(il,11),'%1.1f'),'Color',line_col{il});
```

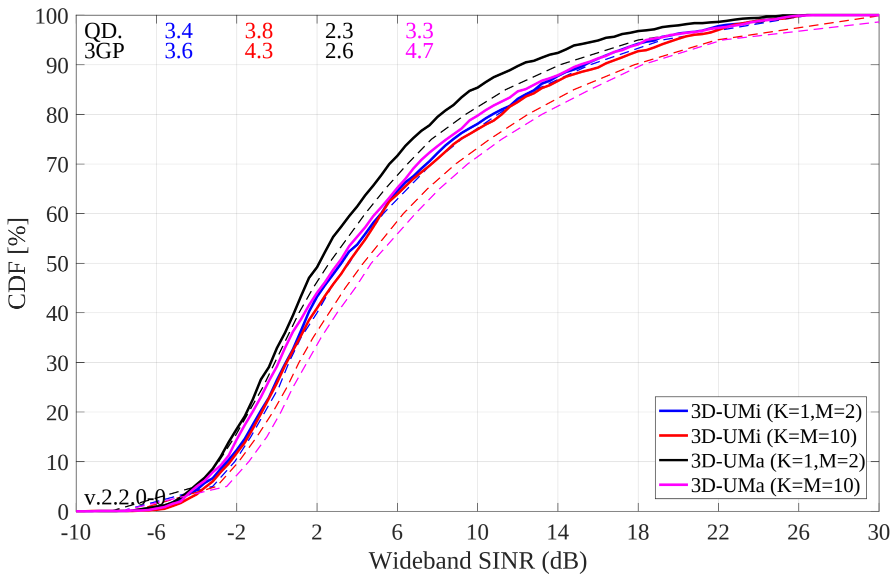
```
end

hold off; grid on; box on;
set(gca,'YTick',0:10:100); set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
xlabel('Coupling loss (dB)')
ylabel('CDF [%]')
legend(ln,legend_names( create_curves ),'Location', 'SouthEast')
text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
```



Wideband SINR The wideband SINR is essentially the same as the GF. However, the 3GPP model uses the RSRP values for the calculation of this metric. The calculation method is described in 3GPP TR 36.873 V12.5.0 in Section 8.1 on Page 38. Essentially, the RSRP values describe the average received power (over all antenna elements at the receiver) for each transmit antenna port. Hence, in the phase 2 calibration, there are 4 RSRP values, one for each transmit antenna. The wideband SINR is the GF calculated from the first RSRP value, i.e. the average power for the first transmit antenna port.

```
% Calculate the RSRP value from the first transmit antenna
rsrp_p0 = zeros( no_rx, 19*3, 4 );
for il = create_curves
    for ir = 1 : no_rx
        for it = 1 : 19*3
            tmp = c(ir,it,il).coeff(:,1,:);                    % Coefficients from first Tx antenna
            rsrp_p0( ir,it,il ) = sum( abs( tmp(:) ).^2 ) / 2;      % Divide by 2 Rx antennas
        end
    end
end

% Calculate wideband SINR
sinr = zeros( no_rx, 4 );
for il = create_curves
    sinr(:,il) = 10*log10( max( rsrp_p0(:,:,il),[],2 ) ./ ...
        ( sum( rsrp_p0(:,:,il),2 ) - max( rsrp_p0(:,:,il),[],2 ) ) );
end

figure('Position',[ 50 , 550 , 950 , 600]);
axes('position',[0.09 0.12 0.88 0.86]); hold on;

xm = -10; wx = 40; tx = 0.01; ty = 97;
text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
ln = []; bins = (-0.1:0.01:1.1)*wx+xm;
for il = create_curves
    ln(end+1) = plot( bins, 100*qf.acdf(sinr(:,il),bins),['-',line_col{il}],'Linewidth',2);
    plot( sinr36873b(il,:), 0:5:100,['--',line_col{il}],'Linewidth',1 )
    text((tx+0.1*il)*wx+xm,ty,num2str(median(sinr(:,il)),'%1.1f'),'Color',line_col{il});
    text((tx+0.1*il)*wx+xm,ty-4,num2str(sinr36873b(il,11),'%1.1f'),'Color',line_col{il});
```

```
end

hold off; grid on; box on;
set(gca,'YTick',0:10:100); set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
xlabel('Wideband SINR (dB)')
ylabel('CDF [%]')
legend(ln,legend_names( create_curves ),'Location', 'SouthEast')
text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
```



Zenith of Departure Spread The zenith of departure spread is calculated without the influence of the antenna patterns. Only the raw value before weighting the path powers with the antenna gain is used. This is not immediately clear from 3GPP TR 36.873, because the calculation method is not specified. However, a high gain pattern, such as used for the K=M=10 cases, would significantly decrease the angular spread compared to the low-gain patterns (K=1, M=2) since may paths get less power due to the weighting with the antenna pattern. Hence, we conclude that the angular spreads are calculated without influence of the antenna patterns. Unfortunately, 3GPP also does not define how the angular spread is calculated. Here, we extract the angles and the path powers from the QuaDRiGa SSF model and calculate the RMS angular spread as

$$\bar{\phi} = \arg\left(\sum_{l=1}^{L} P_l \cdot \exp\left(j\phi_l\right)\right)$$

$$\phi_l^{[*]} = \left(\phi_l - \bar{\phi} + \pi \mod 2\pi\right) - \pi$$

$$\sigma_\phi = \sqrt{\frac{1}{P} \cdot \sum_{l=1}^{L} P_l \cdot \left(\phi_l^{[*]}\right)^2 - \left(\frac{1}{P} \cdot \sum_{l=1}^{L} P_l \cdot \phi_l^{[*]}\right)^2}$$

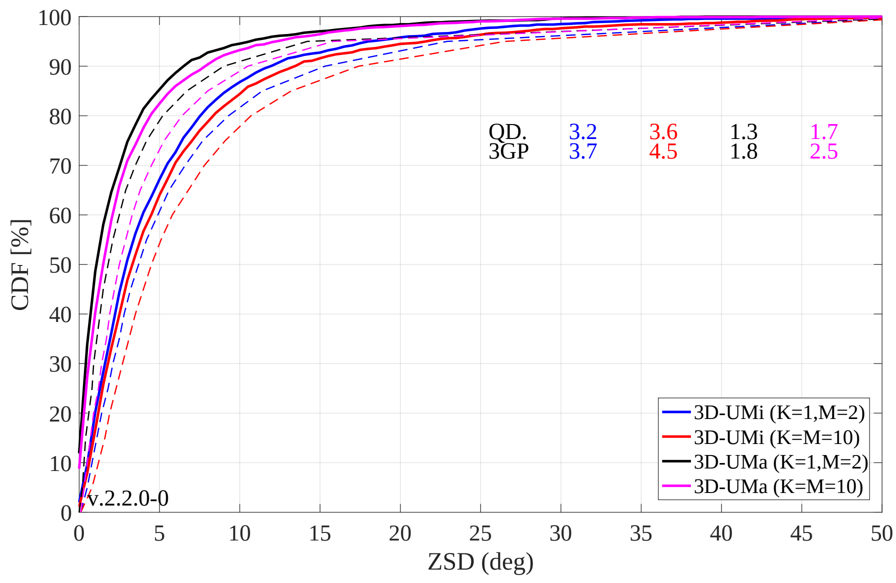
where ϕ_l is the raw departure or arrival angle of a path obtained from the model, $\bar{\phi}$ is the mean angle of all paths belonging to a CIR, and $\phi_l^{[*]}$ is the angle where the mean angle is equal to 0 degree. P_l is the power of a path, P is the total power in the CIR, and L is the number of paths.

To gain some information about the expected values, we can use the formulas in 3GPP TR 36.873, page 37. Most of the users are in NLOS conditions and 80 percent of them are situated indoors. Simulation results show that the average distance between the MT and the serving BS is 0.65 times the ISD. Also, the average height for the indoor users is 9 m. With those values, the expected median ZSD for this case are:

$$\mu_{ZSD}(\text{UMa, NLOS, O2I}) = 10^{-2.1(d_{2D}/1000)-0.01(h_{MT}-1.5)+0.9} \approx 1.4^\circ$$
$$\mu_{ZSD}(\text{UMi, NLOS, O2I}) = 10^{-2.1(d_{2D}/1000)+0.01\cdot\max(h_{MT}-h_{BS},0)+0.9} \approx 4.4^\circ$$

Results in the figure show that the median ZOD values for the 3GPP calibration are around 4 degree for UMi and 2 degree for UMa. However, QuaDRiGa produces smaller values of 3 degree for UMi and 1.7 degree for UMa.

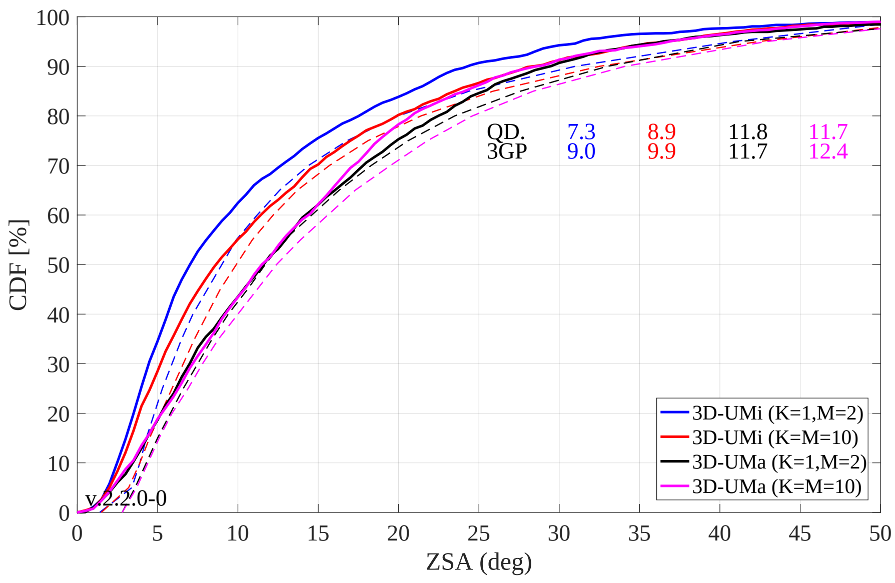
```
zsd = zeros( no_rx, 4 );
for il = create_curves
    for ir = 1 : no_rx
        [~,it] = max(pg_eff(ir,:,il),[],2);                % Determine the serving BS
        eod = c(ir,it,il).par.EoD_cb *pi/180;              % EoD angle in [rad]
        pow = c(ir,it,il).par.pow_cb;                      % Normalized power w/o antenna
        zsd(ir,il) = qf.calc_angular_spreads( eod,pow );   % ZSD = ESD in [rad]
    end
end
zsd = zsd * 180 / pi;                                      % Convert to [deg]

figure('Position',[ 50 , 550 , 950 , 600]);
axes('position',[0.09 0.12 0.88 0.86]); hold on;

xm = 0; wx = 50; tx = 0.51; ty = 77;
text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
ln = []; bins = (-0.1:0.01:1.1)*wx+xm;
for il = create_curves
    ln(end+1) = plot( bins, 100*qf.acdf(zsd(:,il),bins),['-',line_col{il}],'Linewidth',2);
    plot( zsb36873b(il,:), 0:5:100,['--',line_col{il}],'Linewidth',1 )
    text((tx+0.1*il)*wx+xm,ty,num2str(median(zsd(:,il)),'%1.1f'),'Color',line_col{il});
    text((tx+0.1*il)*wx+xm,ty-4,num2str(zsb36873b(il,11),'%1.1f'),'Color',line_col{il});
end

hold off; grid on; box on;
set(gca,'YTick',0:10:100); set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
xlabel('ZSD (deg)')
ylabel('CDF [%]')
legend(ln,legend_names( create_curves ),'Location', 'SouthEast')
text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
```



Zenith of Arrival Spread The ZSA is calculated in the same way as the ZSD. It is notable here, that the for all O2I scenarios, identical values were proposed for the ZSA in 3GPP TR 36.873. The median value is given as 10.2 degree. Since 80 percent of the MTs are indoors, the median value should be around 10.2

for all scenarios and antenna configurations. Surprisingly, the results show differences in the median ZSA, depending on the antenna and scenario settings for both, the 3GPP-3D reference curves and the QuaDRiGa results. The reason for this is currently subject to speculation. As for the ZSD, QuaDRiGa tends to predict slightly lower median values compared to the 3GPP-3D reference.

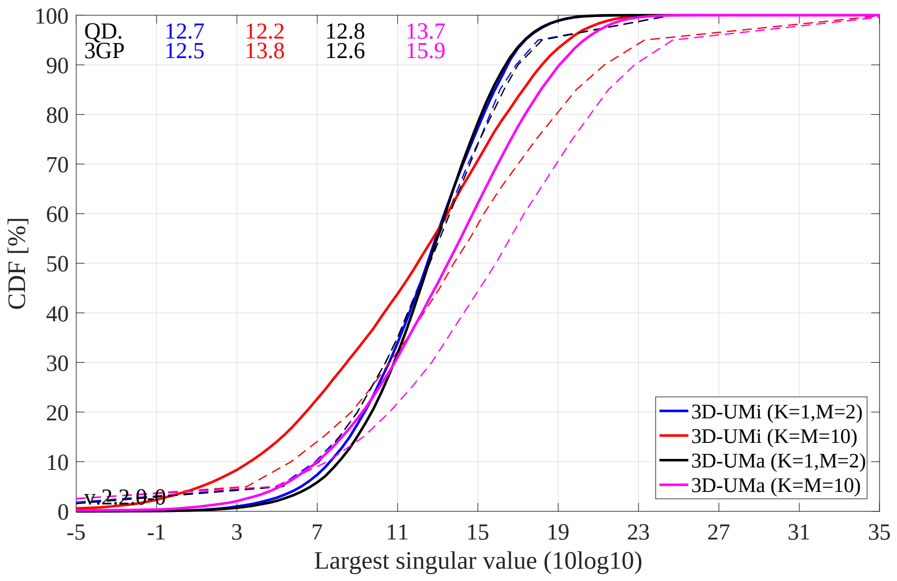
```
zsa = zeros( no_rx , 4 );
for il = create_curves
    for ir = 1 : no_rx
        [~,it] = max(pg_eff(ir,:,il),[],2);                  % Determine the serving BS
        eod = c(ir,it,il).par.EoA_cb *pi/180;                % EoD angle in [rad]
        pow = c(ir,it,il).par.pow_cb;                        % Normalized power w/o antenna
        zsa(ir,il) = qf.calc_angular_spreads( eod,pow );     % ZSD = ESD in [rad]
    end
end
zsa = zsa * 180 / pi;                                        % Convert to [deg]

figure('Position',[ 50 , 550 , 950 , 600]);
axes('position',[0.09 0.12 0.88 0.86]); hold on;

xm = 0; wx = 50; tx = 0.51; ty = 77;
text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
ln = []; bins = (-0.1:0.01:1.1)*wx+xm;
for il = create_curves
    ln(end+1) = plot( bins, 100*qf.acdf(zsa(:,il),bins),['-',line_col{il}],'Linewidth',2);
    plot( zsa36873b(il,:), 0:5:100,['--',line_col{il}],'Linewidth',1 )
    text((tx+0.1*il)*wx+xm,ty,num2str(median(zsa(:,il)),'%1.1f'),'Color',line_col{il});
    text((tx+0.1*il)*wx+xm,ty-4,num2str(zsa36873b(il,11),'%1.1f'),'Color',line_col{il});
end

hold off; grid on; box on;
set(gca,'YTick',0:10:100); set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
xlabel('ZSA (deg)')
ylabel('CDF [%]')
legend(ln,legend_names( create_curves ),'Location', 'SouthEast')
text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
```



Largest and smallest singular values The singular values of a MIMO channel matrix describe how many parallel spatial data streams can be transmitted to one user and what the individual capacity of each streams is. The simulation settings propose two settings: One with four vertically polarized antennas at the BS and two vertically polarized antennas at the receiver (configuration 1), and one with two cross-polarized high-gain an antennas at the BS and an ideal cross-polarized array antenna at the receiver (configuration 2). Both configurations result in a 2x4 MIMO channel. Hence, the channel has two singular values and supports at most two streams. The 3GPP-3D report does not mention, how the singular values are calculated from the channel matrix. It was only discussed internally. The method is as follows:

- The results are reported for the channel matrix of the serving BS. The serving BS is determined at the MT by the highest received power of all BS in the layout.
- The calculations are done in the frequency domain. The bandwidth is set to 10 MHz, which is further split into 50 resource blocks (RBs) of 200 kHz bandwidth, each. Each RB can further be divided inton sub-carriers. However, for the QuaDRiGa results, we only used one subcarrier per RB.
- The singular values are reported for channels without path-gain, but with antenna patterns included. Hence, one needs to extract the path-gain at the MT position from the channel model and % normalize the channel matrix accordingly, i.e.
$$\mathbf{H} = \frac{\mathbf{H}^{[raw]}}{\sqrt{10^{0.1\cdot PG_{dB}}}}$$
- The "singular values" are calculated for each RB by an Eigen-value decomposition of the receive covariance matrix as
$$s_{1,2} = \frac{1}{n_{RB}} \cdot \text{eig}\left(\sum_{n=1}^{n_{RB}} \mathbf{H}_n \mathbf{H}_n^H\right)$$
for one single carrier, the relationship between the eigenvalues of the covariance matrix and the singular values of the channel matrix is given by
$$s_{1,2} = \text{eig}\left(\mathbf{H}_n \mathbf{H}_n^H\right) = \left\{\text{svd}\left(\mathbf{H}\right)\right\}^2$$
- Results are presented in logarithmic scale, i.e. as $10 \cdot \log_{10}(s_{1,2})$.

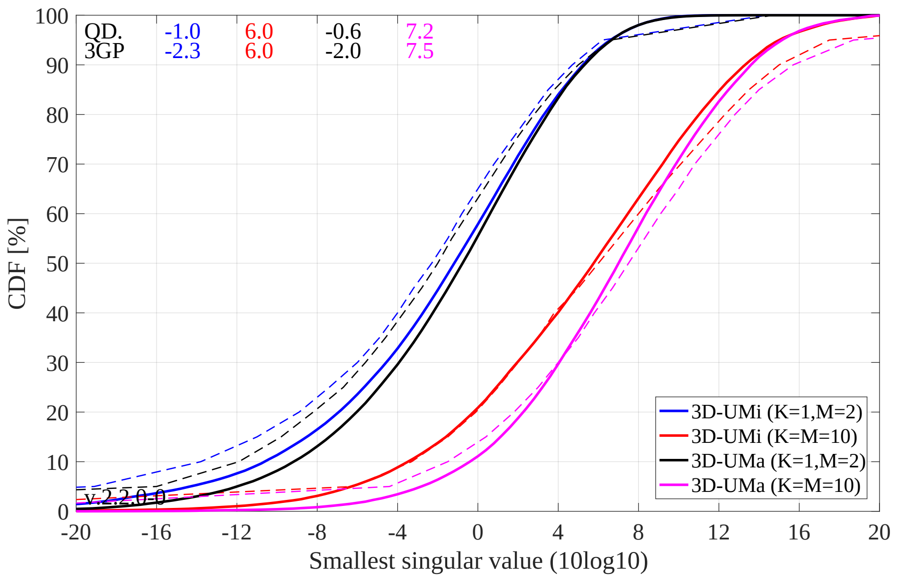
```
sv = zeros( 2,50,no_rx,4 );
for il = create_curves
    for ir = 1 : no_rx
        [~,it] = max(pg_eff(ir,:,il),[],2);                     % Determine the serving BS

        % Frequency-Domain channel matrix @ 50 RBs, 10 MHz
        H = c(ir,it,il).fr( 10e6, 50 );

        % Get the PG without antenna pattern. This is stored in c.par.pg_parset.
        pg = c(ir,it,il).par.pg_parset;                         % in [dB]
        H = H ./ sqrt(10.^(0.1*pg));                            % Normalize channel matrix

        for m = 1:size(H,3)
            sv(:,m,ir,il) = svd(H(:,:,m)).^2;
        end  % NOTE: eig( H(:,:,m)*H(:,:,m)' ) == svd(H(:,:,m)).^2
    end
end

figure('Position',[ 50 , 550 , 950 , 600]);
axes('position',[0.09 0.12 0.88 0.86]); hold on;

xm = -5; wx = 40; tx = 0.01; ty = 97;
text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
ln = []; bins = (-0.1:0.01:1.1)*wx+xm;
for il = create_curves
    sv_max = 10*log10( reshape(sv(1,:,:,il),[],1) );
    ln(end+1) = plot( bins, 100*qf.acdf(sv_max,bins),['-',line_col{il}],'Linewidth',2);
    plot( sv1_36873b(il,:), 0:5:100,['--',line_col{il}],'Linewidth',1 )
    text((tx+0.1*il)*wx+xm,ty,num2str(median(sv_max),'%1.1f'),'Color',line_col{il});
    text((tx+0.1*il)*wx+xm,ty-4,num2str(sv1_36873b(il,11),'%1.1f'),'Color',line_col{il});
end

hold off; grid on; box on;
set(gca,'YTick',0:10:100); set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
xlabel('Largest singular value (10log10)')
ylabel('CDF [%]')
legend(ln,legend_names( create_curves ),'Location', 'SouthEast')
text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
```



The generated figure shows the distribution of the largest singular value. For the results with co-polar antennas (the blue and black curve), there is an almost perfect match between QuaDRiGa and the 3GPP calibration curves. The results for the cross-polar antennas (red and magenta line) show some differences. However, the results from individual partners in R1-143469-2014also show a significant spread in this case. Median results for the UMi scenario (red curve) ranged from 9 to 15 dB. QuaDRiGa predicts 10.6 dB, which is still well within the reported range.

Smallest singular value The results for the smallest singular value are shown in the following figure. Here, QuaDRiGa performs very close to the median results reported in R1-143469-2014.

```
figure('Position',[ 50 , 550 , 950 , 600]);
axes('position',[0.09 0.12 0.88 0.86]); hold on;

xm = -20; wx = 40; tx = 0.01; ty = 97;
text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
ln = []; bins = (-0.1:0.01:1.1)*wx+xm;
for il = create_curves
    sv_min = 10*log10( reshape(sv(2,:,:,il),[],1) );
    ln(end+1) = plot( bins, 100*qf.acdf(sv_min,bins),['-',line_col{il}],'Linewidth',2);
    plot( sv2_36873b(il,:), 0:5:100,['--',line_col{il}],'Linewidth',1 )
    text((tx+0.1*il)*wx+xm,ty,num2str(median(sv_min),'%1.1f'),'Color',line_col{il});
    text((tx+0.1*il)*wx+xm,ty-4,num2str(sv2_36873b(il,11),'%1.1f'),'Color',line_col{il});
end

hold off; grid on; box on;
set(gca,'YTick',0:10:100); set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
xlabel('Smallest singular value (10log10)')
ylabel('CDF [%]')
legend(ln,legend_names( create_curves ),'Location', 'SouthEast')
text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
```

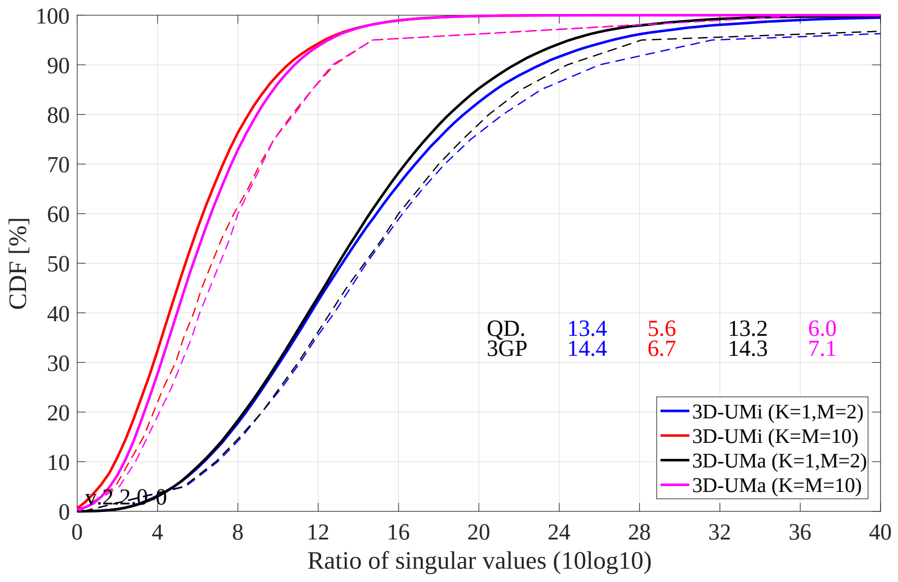


Ratio of singular values Probably a more important measure than the singular values themselves is the ratio between the singular values, which is calculated as

$$SR = 10 \cdot \log_{10}\left(\frac{s_1}{s_2}\right)$$

This measure is closely linked to the condition number of the channel matrix $C = \sqrt{\frac{s_1}{s_2}}$. The larger this number is, the more difficult it is to invert the matrix $\mathbf{H}$. However, inverting this matrix is required in order to separate the two data streams at the receiver.

```
figure('Position',[ 50 , 550 , 950 , 600]);
axes('position',[0.09 0.12 0.88 0.86]); hold on;

xm = 0; wx = 40; tx = 0.51; ty = 37;
text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
ln = []; bins = (-0.1:0.01:1.1)*wx+xm;
for il = create_curves
    sv_rat = 10*log10( reshape( sv(1,:,:,il) ./ sv(2,:,:,il) ,[],1) );
    ln(end+1) = plot( bins, 100*qf.acdf(sv_rat,bins),['-',line_col{il}],'Linewidth',2);
    plot( svR_36873b(il,:), 0:5:100,['--',line_col{il}],'Linewidth',1 )
    text((tx+0.1*il)*wx+xm,ty,num2str(median(sv_rat),'%1.1f'),'Color',line_col{il});
    text((tx+0.1*il)*wx+xm,ty-4,num2str(svR_36873b(il,11),'%1.1f'),'Color',line_col{il});
end

hold off; grid on; box on;
set(gca,'YTick',0:10:100); set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
xlabel('Ratio of singular values (10log10)')
ylabel('CDF [%]')
legend(ln,legend_names( create_curves ),'Location', 'SouthEast')
text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
```



As can be seen, the ratio is much higher for the co-polar antenna configuration (blue and black curve). For cross-polar channels, the ratio is about one order of magnitude lower, since an additional degree of freedom is provided by the second polarization. Results from QuaDRiGa generally agree well. However, there is one exception for the 3D-UMa cross-polar case, where QuaDRiGa predicts a SV-ratio of 6.1 dB. The lowest reported value in R1-143469-2014 is 6.3 dB.

5.3 3GPP 38.901 Large Scale Calibration

This section performs the 3GPP calibration as described in 3GPP TR 38.901 V14.1.0, Section 7.8.1, Page 74 for the large scale calibration. It is shown how the model is set up to obtain the required results, how the output is processed and how the results compare with the 3GPP baseline. The purpose of this calibration is to show the correct working of the path-loss models, the antenna model, the user placement in 3D coordinates.

Antenna setup 3GPP uses a high-gain panel antenna with 10 coupled elements in elevation and 12 degree electric down-tilt for UMa and UMi scenarios. Indoor scenarios use 20 degree downtilt. Note: The 102 or 110 degree electrical tilt in Table 7.8-1 refer to spheric coordinates, whereas QuaDRiGa uses geographic coordinates.

```
close all
clear all

% Antenna configuration 1 (UMa and UMi)
% 10 elements in elevation, 1 element in azimuth, vertical pol., 12 deg downtilt, 0.5 lambda spacing
a1 = qd_arrayant( '3gpp-3d', 10, 1, [], 4, 12, 0.5 );
a1.element_position(1,:) = 0.5;              % Distance from pole in [m]

% Antenna configuration 1 (Indoor)
% 10 elements in elevation, 1 element in azimuth, vertical pol., 20 deg downtilt, 0.5 lambda spacing
a2 = qd_arrayant( '3gpp-3d', 10, 1, [], 4, 20, 0.5 );
a2.element_position(1,:) = 0.2;              % Distance from pole in [m]
```

QuaDRiGa Setup Here, the channel model is configured. The simulation assumptions are given in Table 7.8.1 in 3GPP TR 38.901 V14.1.0. 3GPP specifies to perform simulations for UMa, UMi and Indoor at 3 frequencies: 6 GHz, 30 GHz and 70 GHz. The scenario parameters for UMa and UMi are given in Table 7.2-1, page 20. Hence, we define three QuaDRIGa layouts. UMa and UMi use a hexagonal grid with 19 sites and three sectors per site. This is implemented in "qd_layout.generate", using the "regular" layout. The indoor scenario layout is speciefied in Table 7.2-2.

```
no_rx = 2000;                               % Number of MTs (directly scales the simulation time)
select_scenario = 1:3;                      % Scenario: 1 = UMi, 2 = UMa, 3 = Indoor
select_fequency = 1:3;                      % Frequency: 1 = 6 GHz, 2 = 30 GHz, 4 = 70 GHz

s = qd_simulation_parameters;               % Set general simulation parameters
s.center_frequency = [ 6e9, 30e9, 70e9 ];   % Set center frequencies for the simulations
s.center_frequency = s.center_frequency( select_fequency );
no_freq = numel( s.center_frequency );

s.use_3GPP_baseline = 1;                    % Disable spherical waves
s.show_progress_bars = 0;                   % Disable progress bar

isd = [ 200, 500, 20 ];                                         % ISD in each layout
no_go_dist = [ 10, 35, 0 ];                                     % Min. UE-eNB 2D distance

l(1,1) = qd_layout.generate( 'regular', 19, isd(1), a1);
l(1,1).simpar = s;                                              % Set simulation parameters
l(1,1).tx_position(3,:) = 10;                                   % 10 m BS height
l(1,1).name = 'UMi';

l(1,2) = qd_layout.generate( 'regular', 19, isd(2), a1);
l(1,2).tx_position(3,:) = 25;                                   % 25 m BS height
l(1,2).simpar = s;                                              % Set simulation parameters
l(1,2).name = 'UMa';

l(1,3) = qd_layout.generate( 'indoor', [2,6], isd(3), a2, 3, 30);
l(1,3).tx_position(3,:) = 3;                                    % 3 m BS height
l(1,3).simpar = s;                                              % Set simulation parameters
l(1,3).name = 'Indoor Open Office';

for il = select_scenario                                        % Dorp users in each layout
    l(1,il).no_rx = no_rx;                                      % Number of users
    if il == 3
        ind = true( 1,no_rx );                                  % Indoor placement
        while any( ind )
            l(1,il).randomize_rx_positions( sqrt(60^2+25^2), 1, 1, 0, ind );
            ind = abs( l(1,il).rx_position(1,:) ) > 60 | abs( l(1,il).rx_position(2,:) ) > 25;
        end
    else
        ind = true( 1,no_rx );                                  % UMa / UMi placement
        while any( ind )
            l(1,il).randomize_rx_positions( 0.93*isd(il), 1.5, 1.5, 0, ind );
            ind = sqrt(l(1,il).rx_position(1,:).^2 + l(1,il).rx_position(2,:).^2) < no_go_dist(il);
        end
        floor = randi(5,1,l(1,il).no_rx) + 3;                   % Number of floors in the building
        for n = 1 : l(1,il).no_rx
            floor( n ) =  randi(  floor( n ) );                 % Floor level of the UE
        end
        l(1,il).rx_position(3,:) = 3*(floor-1) + 1.5;           % Height in meters
    end
    switch il   % Set the scenario and assign LOS probabilities (80% of the users are indoor)
        case 1
            indoor_rx = l(1,il).set_scenario('3GPP_38.901_UMi',[],[],0.8);
            l(1,il).rx_position(3,~indoor_rx) = 1.5;            % Set outdoor-users to 1.5 m height
        case 2
            indoor_rx = l(1,il).set_scenario('3GPP_38.901_UMa',[],[],0.8);
            l(1,il).rx_position(3,~indoor_rx) = 1.5;            % Set outdoor-users to 1.5 m height
        case 3
            l(1,il).set_scenario('3GPP_38.901_Indoor_Open_Office');
    end
    l(1,il).rx_array = qd_arrayant('omni');                     % Omni-Antenna, vertically polarized
end
```

```
Warning: Multi-frequency simulations are not compatible with the 3GPP baseline
model. Path parameters will be uncorrellated.
```

Generate channels The following code generates the channel coefficients. However, by default, QuaDRiGa always uses the full small-scale-fading model as well as spatial consistency. These two features are disabled by setting the number of paths to 1 and the decorrelation distance for the SSF (SC_lambda) to 0 m.

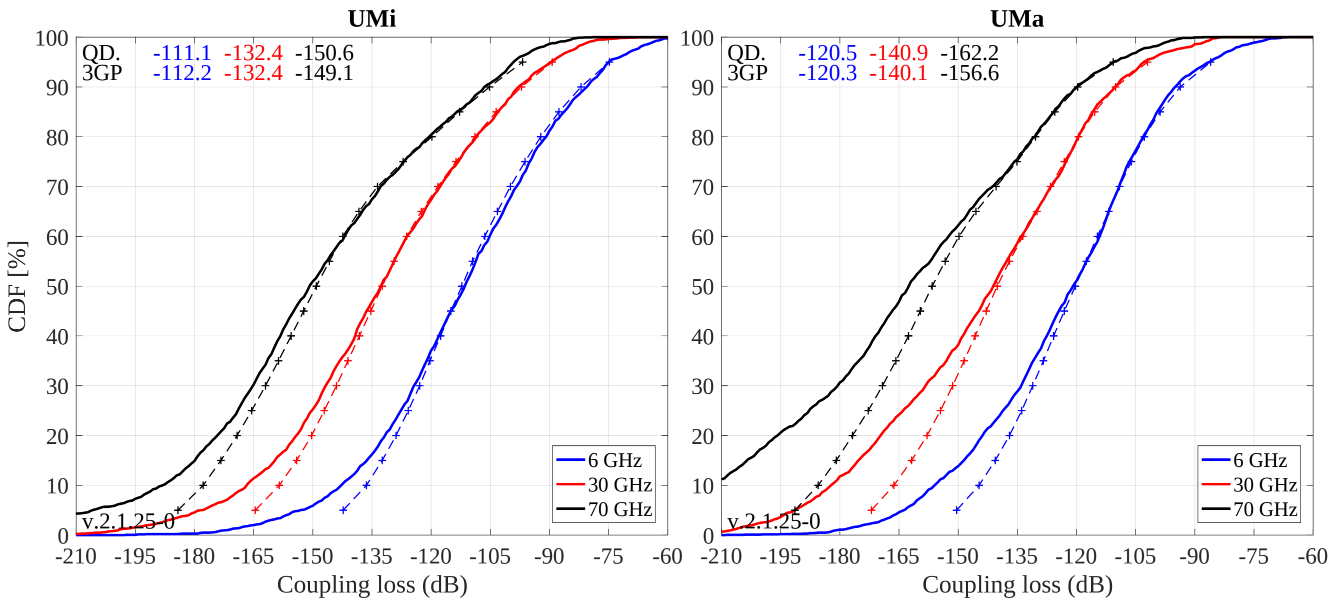
```
tic
pg_eff = cell( 1,3 );
for il = select_scenario
    pg_eff{il} = zeros( no_rx , l(1,il).no_tx*3 , no_freq );
    b = l(1,il).init_builder;                          % Generate builders

    sic = size( b );
    for ib = 1 : numel(b)
        [ i1,i2 ] = qf.qind2sub( sic, ib );
        scenpar = b(i1,i2).scenpar;                    % Read scenario parameters
        scenpar.NumClusters = 1;                       % Only LOS path, disable SSF model
        scenpar.SC_lambda = 0;                         % Disable spatial consistency of SSF
        b(i1,i2).scenpar_nocheck = scenpar;            % Save parameters without check (faster)
    end

    b = split_multi_freq( b );                         % Split the builders for multiple frequencies
    gen_parameters( b );                               % Generate LSF (SF) and SSF (LOS path only)
    cm = get_channels( b );                            % Generate channels
    cm = split_tx( cm, {1,2,3} );                      % Split sectors
    cm = qf.reshapeo( cm, [ no_rx, l(1,il).no_tx*3, no_freq ] );

    for ir = 1 : no_rx                                 % Extract effective PG vor each BS-MT link
        for it = 1 : l(1,il).no_tx*3
            for iF = 1 : no_freq
                pg_eff{il}( ir,it,iF ) = abs( cm( ir,it,iF ).coeff ).^2;
            end
        end
    end
end
toc
```

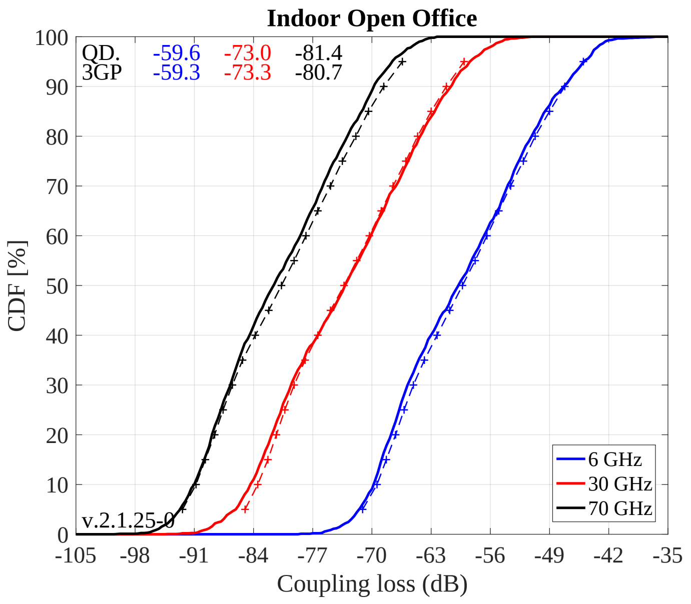
```
Elapsed time is 1294.425458 seconds.
```

Coupling Loss The coupling loss is defined as the path gain of a MT to its serving BS, i.e. the strongest BS seen by the MT. The thick lines were obtained using the QuaDRiGa model, the thin dashed line are taken from 3GPP R1-165974. They represent the median of all 3GPP calibration results. Results agree well for UMi and Indoor Open Office. However, there are some significant differences in the UMa calibration curves. This is probably due to the fact that the original calibration was done using the parameters from 3GPP 38.900 v14.0.0 (2016-06). The parameters for UMa-LOS have changed in 3GPP 38.901 v14.1.0 (2017-06).

```
calib_3GPP_ref_data;                                                % Load reference data

set(0,'defaultTextFontSize', 18)                                    % Default Font Size
set(0,'defaultAxesFontSize', 18)                                    % Default Font Size
set(0,'defaultAxesFontName','Times')                                % Default Font Type
set(0,'defaultTextFontName','Times')                                % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')                      % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')                          % Default Paper Type
set(0,'DefaultFigurePaperSize',[14.5 6.6])                          % Default Paper Size

legend_names = { '6 GHz','30 GHz','70 GHz' };
line_col = {'b','r','k'};                                           % Color of the lines

figure('Position',[ 50 , 550 , 1400 , 640]);
for il = select_scenario
    cl = zeros( no_rx, 3 );              % Calculate the coupling loss from the effective PG
    for iF = 1 : no_freq
        cl(:,iF) = 10*log10(max( pg_eff{il}(:,:,iF),[],2 ));
    end
    if il == 3
        figure('Position',[ 50 , 550 , 1400 , 640]);
        axes('position',[0.3, 0.12 0.44 0.81]); hold on;
        xm = -105; wx = 70; tx = 0.01; ty = 97;
    else
        xm = -210; wx = 150; tx = 0.01; ty = 97;
        axes('position',[0.06+(il-1)*0.48 0.12 0.44 0.81]); hold on;
    end
    text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
    ln = []; bins = (-0.1:0.005:1.1)*wx+xm;
    for iF = 1 : no_freq
```

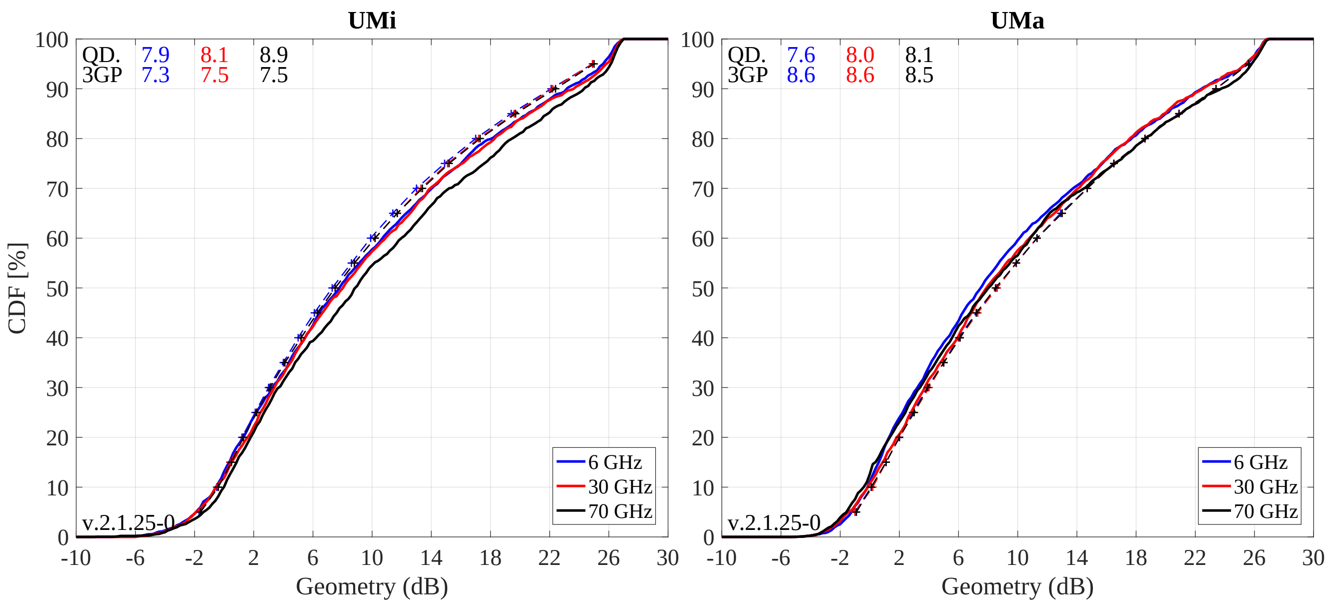
```
        iFs = select_fequency(iF);
        ln(end+1) = plot( bins, 100*qf.acdf(cl(:,iF),bins),['-',line_col{iFs}],'Linewidth',2);
        plot( cl38900a(iFs,:,il), 5:5:95,['+--',line_col{iFs}],'Linewidth',1 )
        text((tx+0.12*iF)*wx+xm,ty,num2str(median(cl(:,iF)),'%1.1f'),'Color',line_col{iFs});
        text((tx+0.12*iF)*wx+xm,ty-4,num2str(cl38900a(iFs,10,il),'%1.1f'),'Color',line_col{iFs});
    end
    hold off; grid on; box on; set(gca,'YTick',0:10:100);
    set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
    xlabel('Coupling loss (dB)')
    if il==1 || il==3; ylabel('CDF [%]'); end;
    title(l(1,il).name)
    legend(ln,legend_names(select_fequency),'Location', 'SouthEast')
    text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
end
```

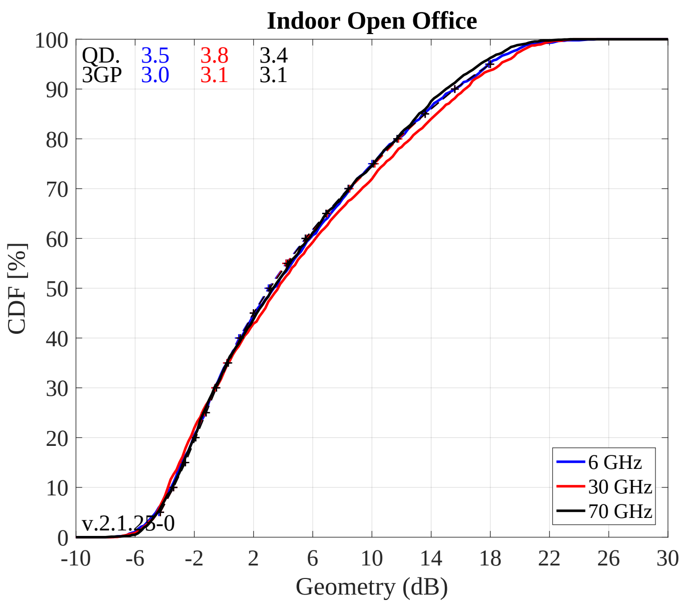




Geometry Factor The GF is a lower bound for the actual SINR. It is defined as the power ratio of the serving BS and the sum power of all interfering BSs. The results in the following Figure agree well with the 3GPP calibrations curves.

```
figure('Position',[ 50 , 550 , 1400 , 640]);
for il = select_scenario
    gf = zeros( no_rx , 3 );
    for iF = 1 : no_freq
        gf(:,iF) = 10*log10( max( pg_eff{il}(:,:,iF),[],2 ) ./ ( sum( pg_eff{il}(:,:,iF),2 ) -...
            max( pg_eff{il}(:,:,iF),[],2 ) ) );
    end
    if il == 3
        figure('Position',[ 50 , 550 , 1400 , 640]);
        axes('position',[0.3, 0.12 0.44 0.81]); hold on;
    else
        axes('position',[0.06+(il-1)*0.48 0.12 0.44 0.81]); hold on;
    end
    xm = -10; wx = 40; tx = 0.01; ty = 97;
    text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
    ln = []; bins = (-0.1:0.005:1.1)*wx+xm;
    for iF = 1:no_freq
        iFs = select_fequency(iF);
        ln(end+1) = plot( bins, 100*qf.acdf(gf(:,iF),bins),['-',line_col{iFs}],'Linewidth',2);
        plot( gf38900a(iFs,:,il), 5:5:95,['+--',line_col{iFs}],'Linewidth',1 )
        text((tx+0.1*iF)*wx+xm,ty,num2str(median(gf(:,iF)),'%1.1f'),'Color',line_col{iFs});
        text((tx+0.1*iF)*wx+xm,ty-4,num2str(gf38900a(iFs,10,il),'%1.1f'),'Color',line_col{iFs});
    end
    hold off; grid on; box on; set(gca,'YTick',0:10:100);
    set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
    xlabel('Geometry (dB)')
    if il==1 || il==3; ylabel('CDF [%]'); end;
    title(l(1,il).name)
    legend(ln,legend_names(select_fequency),'Location', 'SouthEast')
    text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
end
```





5.4 3GPP 38.901 Full Calibration

This section performs the 3GPP calibration as described in 3GPP TR 38.901 V14.1.0, Section 7.8.2, Page 75 for the full calibration. It is shown how the model is set up to obtain the required results, how the output is processed and how the results compare with the 3GPP baseline. The 3GPP calibration reference results were published in the TDOC R1-165975 in August 2016. These results were obtained using model parameters from 3GPP TR 38.900 v14.0.0 (2016-06). Unfortunately, some parameters were changed in the year following the the publication of the results and therefore, different calibration results will be obtained when using the parameters from 38.901 V14.1.0 which are included in QuaDRiGa.

Antenna setup 3GPP uses a nested panel antenna. One panel consists of 16 dual-polarized antenna elements (+/- 45 degree polarization) with 0.5 lambda element spacing. The panel is duplicated along the y-axis. In order to reuse the same antenna object for all four frequencies in the simulation, we do not specify a carrier frequency. In this case the model assumes that the element positions are given in multiples of the wavelength. The method "combine_pattern" calculates the array response with respect to the phase-center of the antenna, i.e. the phase in the antenna pattern then includes the element positions in the array. The effective element positions are set to 0 and the same same antenna can be used for multiple frequencies.

```
close all
clear all

% The mapping function of antenna elements to CRS port (0 degree panning angle)
port_mapping = [ 1,0;0,1 ; 1,0;0,1 ;1,0;0,1 ;1,0;0,1 ];
port_mapping = [ port_mapping , zeros( 8,2 ) ; zeros( 8,2 ), port_mapping ] / 2;

% BS antenna configuration 1 (UMa and UMi), 12 degree downtilt
aBS  = qd_arrayant( '3gpp-mmw', 4, 4, [], 6, 12, 0.5, 1, 2, 2.5, 2.5 );
aBS.coupling = port_mapping;                    % Assign port mapping
aBS.combine_pattern;                            % Calculate array response
aBS.element_position(1,:) = 0.5;               % Distance from pole in [m]

% BS antenna configuration 1 (Indoor), 20 degree downtilt
aBSi  = qd_arrayant( '3gpp-mmw', 4, 4, [], 6, 20, 0.5, 1, 2, 2.5, 2.5 );
aBSi.coupling = port_mapping;                   % Assign port mapping
aBSi.combine_pattern;                           % Calculate array response
aBSi.element_position(1,:) = 0.2;              % Distance from pole in [m]

% BS antenna configuration 2 (UMa, UMi, Indoor)
a2 = qd_arrayant( '3gpp-3d', 2, 2, [], 1, [], 0.5 );
a2.combine_pattern;                             % Calculate array response
```

```
a2.element_position(1,:) = 0.5;                % Distance from pole in [m]

append_array( aBS ,a2 );                       % Concatenate arrays for both configurations
append_array( aBSi,a2 );

aMT = qd_arrayant('omni');                     % MT antenna configuration
aMT.copy_element(1,2);
aMT.Fa(:,:,2) = 0;
aMT.Fb(:,:,2) = 1;
```

QuaDRiGa Setup Here, the channel model is configured. The simulation assumptions are given in Table 7.8-2 in 3GPP TR 38.901 V14.1.0. 3GPP specifies to perform simulations for UMa, UMi and Indoor at four frequencies: 6 GHz, 30 GHz, 60 GHz and 70 GHz. Hence, we define three QuaDRIGa layouts. UMa and UMi use a hexagonal grid with 19 sites and three sectors per site. The scenario parameters for UMa and UMi are given in Table 7.2-1, page 20. This is implemented in "qd_layout.generate", using the "regular" layout. The indoor scenario layout is specified in Table 7.2-2 and implemented using the "indoor" layout. 3GPP defines two different approaches for the LOS probability for InH users (page 27). Here we assume that "open office" should be used.

```
no_rx = 2000;                                  % Number of MTs (directly scales the simulation time)
select_scenario = 1:3;                         % Scenario: 1 = UMi, 2 = UMa, 3 = Indoor
select_fequency = 1:4;                         % Freq.: 1 = 6 GHz, 2 = 30 GHz, 3 = 60 GHz, 4 = 70 GHz

s = qd_simulation_parameters;                  % Set general simulation parameters
s.center_frequency = [ 6e9, 30e9, 60e9, 70e9 ];    % Set center frequencies for the simulations
s.center_frequency = s.center_frequency( select_fequency );
no_freq = numel( s.center_frequency );

s.use_3GPP_baseline = 1;                       % Disable spherical waves
s.show_progress_bars = 0;                      % Disable progress bar

isd = [ 200, 500, 20 ];                                             % ISD in each layout
no_go_dist = [ 10, 35, 0 ];                                         % Min. UE-eNB 2D distance

l(1,1) = qd_layout.generate( 'regular', 19, isd(1), aBS);
l(1,1).simpar = s;                                                  % Set simulation parameters
l(1,1).tx_position(3,:) = 10;                                       % 10 m BS height
l(1,1).name = 'UMi';

l(1,2) = qd_layout.generate( 'regular', 19, isd(2), aBS);
l(1,2).tx_position(3,:) = 25;                                       % 12 m BS height
l(1,2).simpar = s;                                                  % Set simulation parameters
l(1,2).name = 'UMa';

l(1,3) = qd_layout.generate( 'indoor', [2,6], isd(3), aBSi, 3, 30);
l(1,3).tx_position(3,:) = 3;                                        % 3 m BS height
l(1,3).simpar = s;                                                  % Set simulation parameters
l(1,3).name = 'InH';

for il = select_scenario                                            % Drop users in each layout
    l(1,il).no_rx = no_rx;                                          % Number of users
    if il == 3
        ind = true( 1,no_rx );                                      % Indoor placement
        while any( ind )
            l(1,il).randomize_rx_positions( sqrt(60^2+25^2), 1, 1, 0, ind );
            ind = abs( l(1,il).rx_position(1,:) ) > 60 | abs( l(1,il).rx_position(2,:) ) > 25;
        end
    else
        ind = true( 1,no_rx );                                      % UMa / UMi placement
        while any( ind )
            l(1,il).randomize_rx_positions( 0.93*isd(il), 1.5, 1.5, 0, ind );
            ind = sqrt(l(1,il).rx_position(1,:).^2 + l(1,il).rx_position(2,:).^2) < no_go_dist(il);
        end
        floor = randi(5,1,l(1,il).no_rx) + 3;                       % Number of floors in the building
        for n = 1 : l(1,il).no_rx
            floor( n ) =  randi(  floor( n ) );                     % Floor level of the UE
        end
        l(1,il).rx_position(3,:) = 3*(floor-1) + 1.5;               % Height in meters
    end
    switch il   % Set the scenario and assign LOS probabilities (80% of the users are indoor)
```

```
        case 1
            indoor_rx = l(1,il).set_scenario('3GPP_38.901_UMi',[],[],0.8);
            l(1,il).rx_position(3,~indoor_rx) = 1.5;            % Set outdoor-users to 1.5 m height
        case 2
            indoor_rx = l(1,il).set_scenario('3GPP_38.901_UMa',[],[],0.8);
            l(1,il).rx_position(3,~indoor_rx) = 1.5;            % Set outdoor-users to 1.5 m height
        case 3
            l(1,il).set_scenario('3GPP_38.901_Indoor_Open_Office');
    end
    l(1,il).rx_array = aMT;                                     % MT antenna setting
end
```

```
Warning: Multi-frequency simulations are not compatible with the 3GPP baseline
model. Path parameters will be uncorrelated.
```

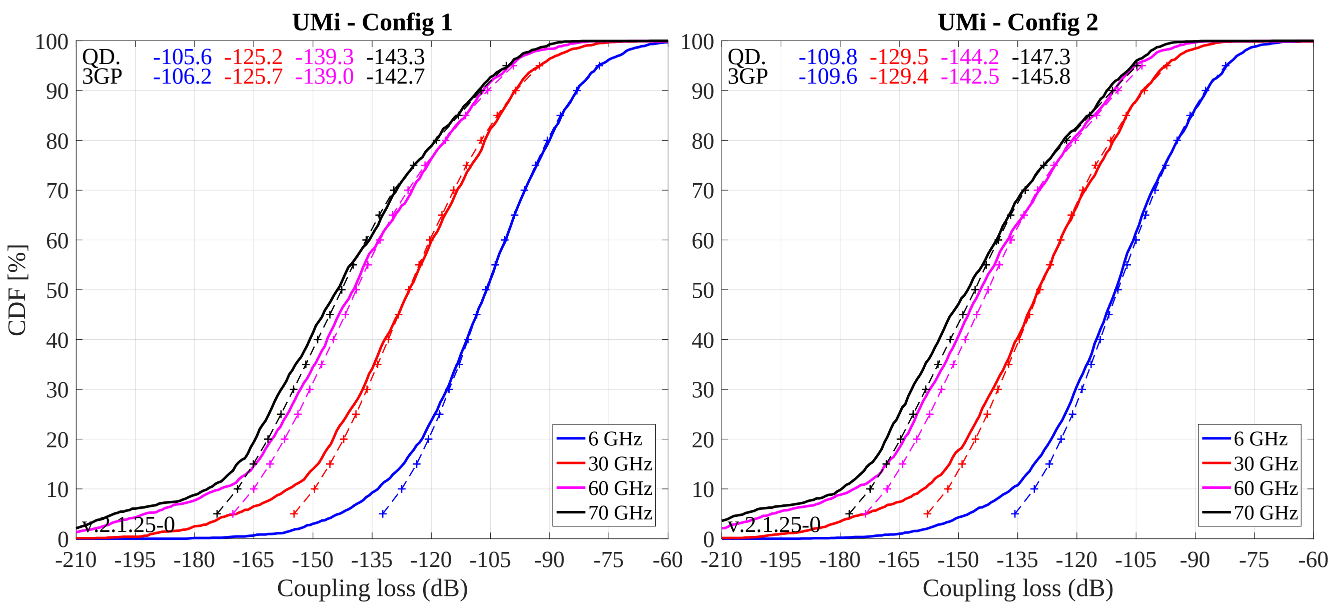
Generate channels The following code generates the channel coefficients. The calibration case assumes that no spatial consistency for the SSF is used. Hence, we deactivate the feature by setting the decorrelation distance of the SSF parameters ”SC_lambda” to 0. The method ”split_multi_freq” separates the builder objects so that each builder creates channels for only one frequency. If you call ”split_multi_freq” before any LSF and SSF parameters are generated as it is done the the following code, then LSF parameters (e.g. SF, DS, AS) will be uncorrelated for each frequency. If you call ”gen_lsf_parameters” before ”split_multi_freq”, then all LSF parameters will be fully correlated. However, frequency dependent averages and variances still apply. If you call ”gen_lsf_parameters” and ”gen_ssf_parameters” before ”split_multi_freq”, then SSF will also correlated, i.e. the same paths will be seen at each frequency. Correlated SSF for multi-frequency simulations is an additional feature of the 3GPP model (see Section 7.6.5, pp 57 of TR 38.901 V14.1.0).

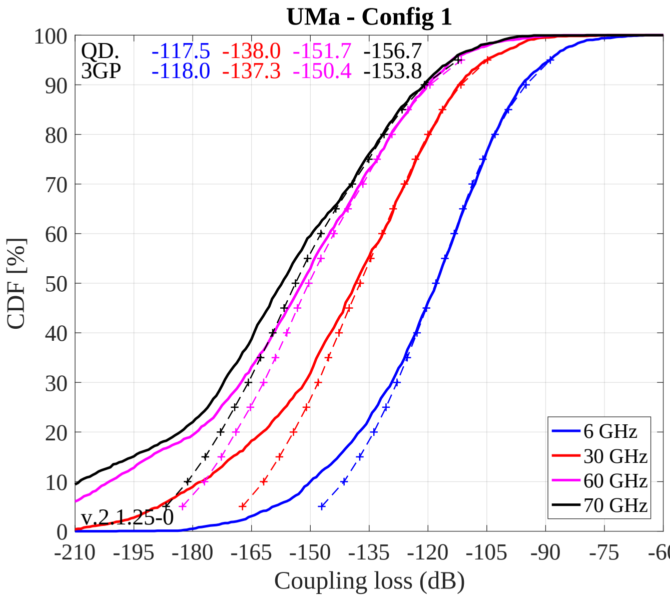
```
tic
clear c
for il = select_scenario
    b = l(1,il).init_builder;                           % Generate builders

    sic = size( b );
    for ib = 1 : numel(b)
        [ i1,i2 ] = qf.qind2sub( sic, ib );
        scenpar = b(i1,i2).scenpar;                     % Read scenario parameters
        scenpar.SC_lambda = 0;                          % Disable spatial consistency of SSF
        b(i1,i2).scenpar_nocheck = scenpar;             % Save parameters without check (faster)
    end

    b = split_multi_freq( b );                          % Split the builders for multiple frequencies
    gen_parameters( b );                                % Generate LSF and SSF parameters (uncorrelated)
    cm = get_channels( b );                             % Generate channels

    cs = split_tx( cm, {1:4,9:12,17:20} );              % Split sectors for Antenna configuration 1
    c{1,il} = qf.reshapeo( cs, [ no_rx, l(1,il).no_tx*3, no_freq ] );
    cs = split_tx( cm, {5:8,13:16,21:24} );             % Split sectors for Antenna configuration 2
    c{2,il} = qf.reshapeo( cs, [ no_rx, l(1,il).no_tx*3, no_freq ] );
end
toc
```

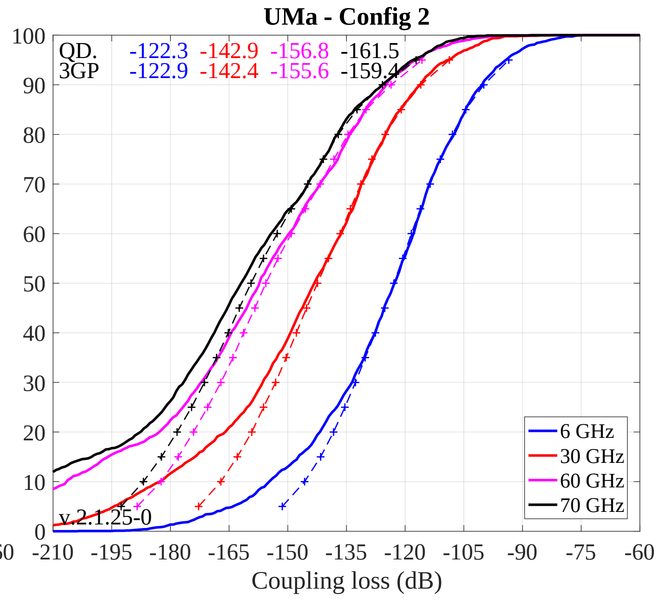
```
Elapsed time is 7775.092647 seconds.
```

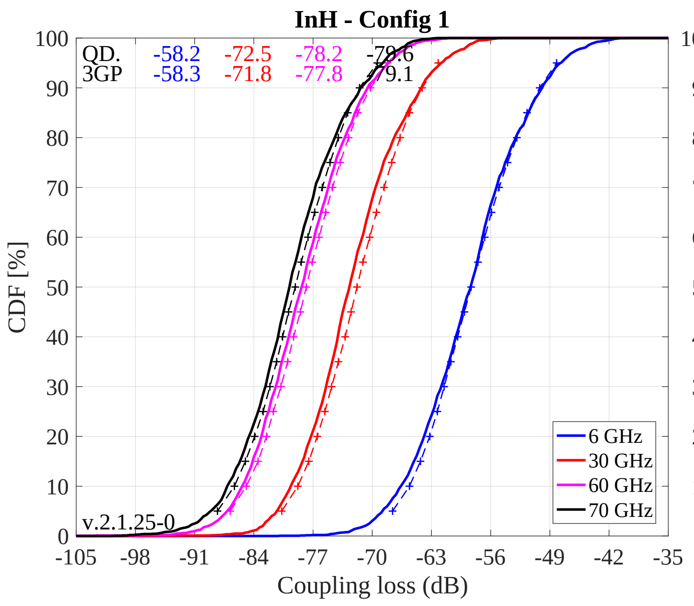
Coupling Loss The coupling loss is defined as the path gain of a MT to its serving BS, i.e. the strongest BS seen by the MT. The thick lines were obtained using the QuaDRiGa model, the thin dashed line are taken from 3GPP R1-165975. They represent the median of all 3GPP calibration results. Results agree well for UMi and Indoor Open Office. However, there are some differences in the UMa calibration curves. This is due to the fact that the 3GPP reference curves were generated using parameters from 3GPP 38.900 v14.0.0 (2016-06). The parameters for UMa-LOS have changed in 3GPP 38.901 v14.1.0 (2017-06).

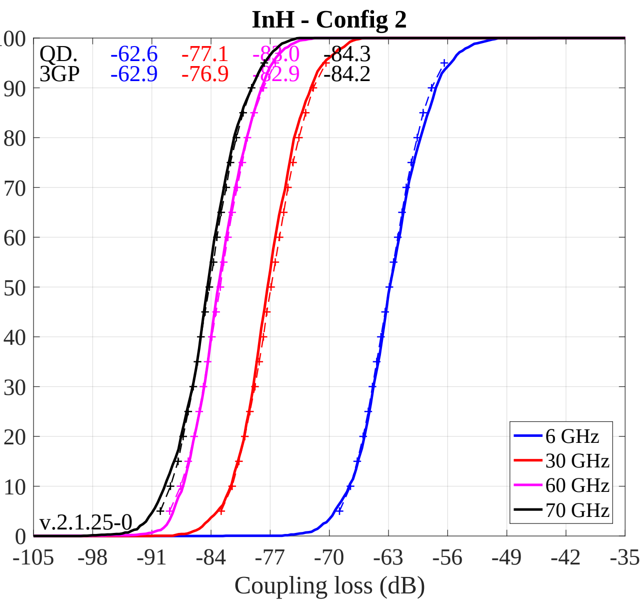
```
calib_3GPP_ref_data;                                              % Load reference data

set(0,'defaultTextFontSize', 18)                                  % Default Font Size
set(0,'defaultAxesFontSize', 18)                                  % Default Font Size
set(0,'defaultAxesFontName','Times')                              % Default Font Type
set(0,'defaultTextFontName','Times')                              % Default Font Type
set(0,'defaultFigurePaperPositionMode','auto')                           % Default Plot position
set(0,'DefaultFigurePaperType','<custom>')                               % Default Paper Type
set(0,'DefaultFigurePaperSize',[14.5 6.6])                               % Default Paper Size

legend_names = { '6 GHz','30 GHz','60 GHz','70 GHz' };
line_col = {'b','r','m','k'};                                            % Color of the lines

for il = select_scenario                                                 % Scenario
    figure('Position',[ 50 , 550 , 1400 , 640]);                         % New figure
    pg_eff = []; cl = [];                                                % Clear variables
    if il  < 3; xm = -210; wx = 150; tx = 0.01; ty = 97; end;            % UMa and UMi
    if il == 3; xm = -105; wx =  70; tx = 0.01; ty = 97; end;            % InH
    for ic = 1:2                                                         % Configuration
        axes('position',[0.06+(ic-1)*0.48 0.12 0.44 0.81]); hold on;     % New sub-figure
        text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');            % Result text
        ln = []; bins = (-0.1:0.005:1.1)*wx+xm;
        for iF = 1 : no_freq                                             % Frequency
            for ir = 1 : no_rx                                           % Receiver
                for it = 1 : size(c{ic,il},2)                            % Calc. path gain
                    pg_eff( it ) = sum(abs(c{ic,il}(ir,it,iF).coeff(:)).^2) / 8;
                end
                cl(ir) = 10*log10(max( pg_eff ));                        % Calc. coupling loss
            end
            iFs = select_fequency(iF); tX = (tx+0.12*iF)*wx+xm;
            ln(end+1) = plot( bins, 100*qf.acdf(cl,bins),['-',line_col{iFs}],'Linewidth',2);
            plot( cl38900b(iFs,:,il,ic), 5:5:95,['+--',line_col{iFs}],'Linewidth',1 )
            text( tX,ty,   num2str(median(cl)             ,'%1.1f'),'Color',line_col{iFs});
            text( tX,ty-4, num2str(cl38900b(iFs,10,il,ic) ,'%1.1f'),'Color',line_col{iFs});
        end
        hold off; grid on; box on; set(gca,'YTick',0:10:100);            % Decorations
        set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
        xlabel('Coupling loss (dB)'); title([ l(1,il).name,' - Config ',num2str( ic ) ] );
        if ic==1; ylabel('CDF [%]'); end;
        legend(ln,legend_names(select_fequency),'Location', 'SouthEast');
        text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
    end
end
```





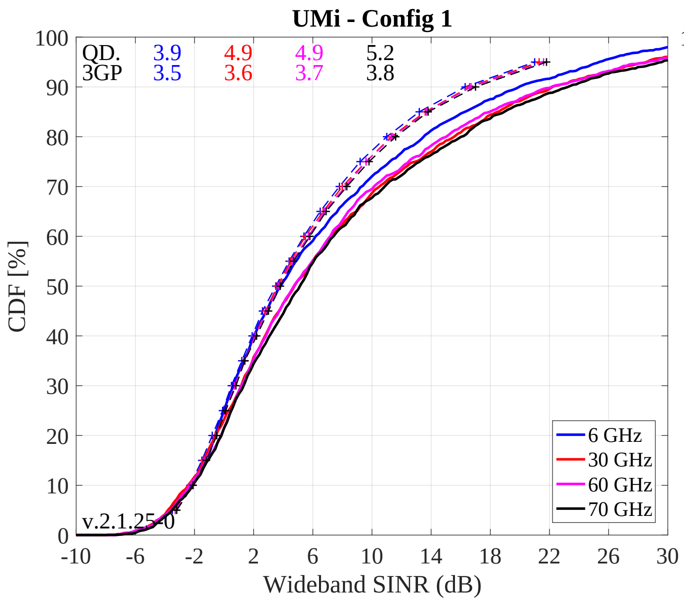


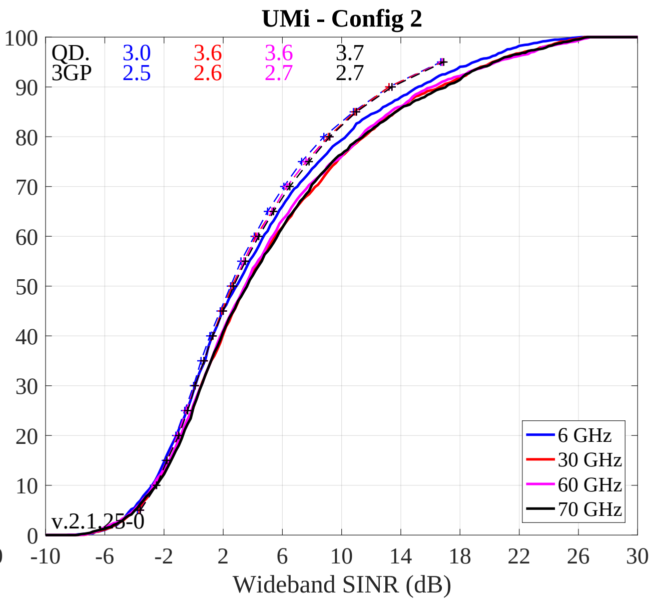


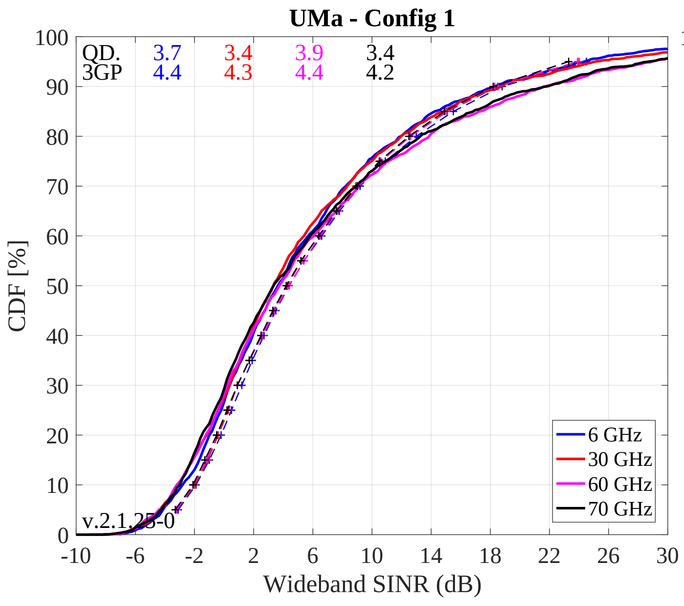


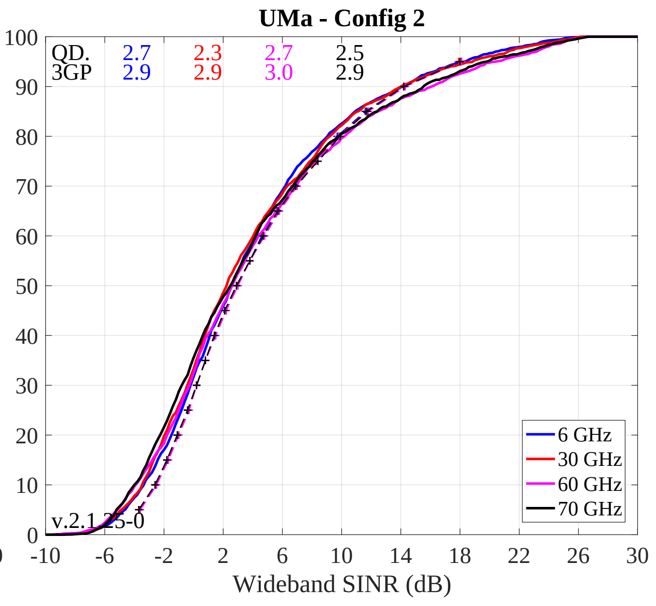
Wide-band SINR The wide-band SINR is essentially the same as the GF. However, the 3GPP model uses the RSRP values for the calculation of this metric. The calculation method is described in 3GPP TR 36.873 V12.5.0 in Section 8.1 on Page 38. Essentially, the RSRP values describe the average received power (over all antenna elements at the receiver) for each transmit antenna port. Hence, there are 4 RSRP values, one for each transmit antenna. The wideband SINR is the GF calculated from the first RSRP value, i.e. the average power for the first transmit antenna port.

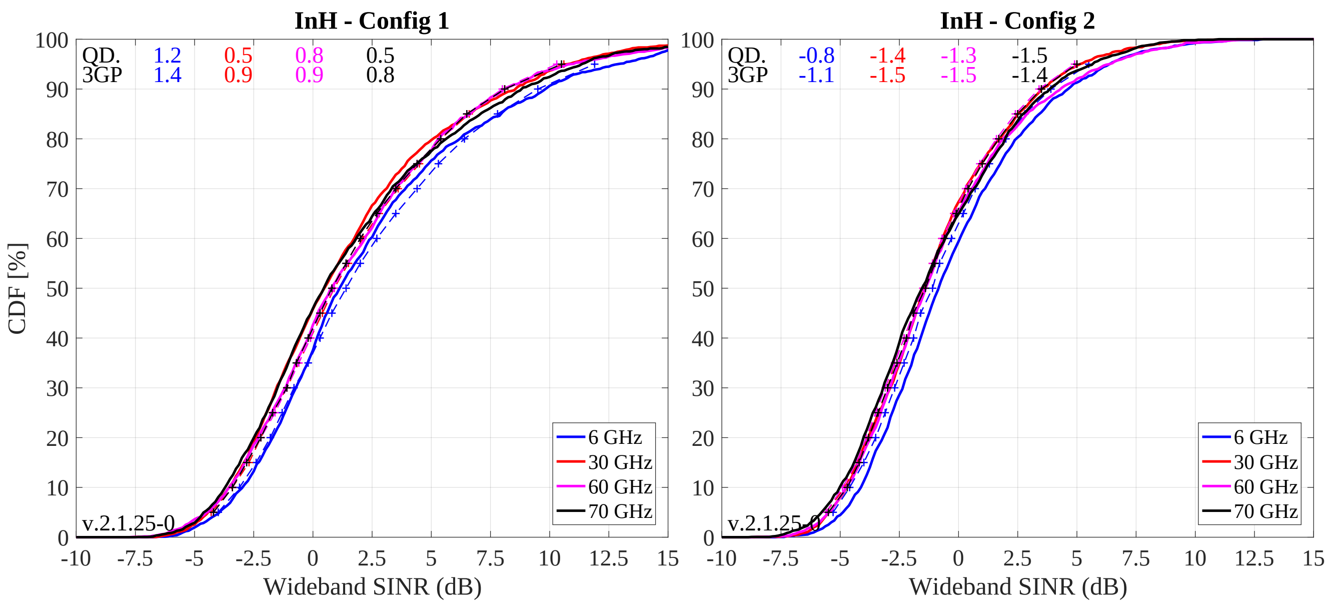
```
for il = select_scenario                                                    % Scenario
    figure('Position',[ 50 , 550 , 1400 , 640]);
    rsrp_p0 = []; cl = [];
    if il==1 || il==2;   xm = -10; wx = 40; tx = 0.01; ty = 97; end;
    if il==3;            xm = -10; wx = 25; tx = 0.01; ty = 97; end;
    for ic = 1 : 2                                                          % Configuration
        axes('position',[0.06+(ic-1)*0.48 0.12 0.44 0.81]); hold on;
        text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
        ln = []; bins = (-0.1:0.005:1.1)*wx+xm;
        for iF = 1 : no_freq                                                % Frequency
            for ir = 1 : no_rx                                              % Calc. coupling loss
                for it = 1 : size(c{ic,il},2)
                    tmp = c{ic,il}(ir,it,iF).coeff(:,1,:);                  % Coeff. from first Tx ant.
                    rsrp_p0( it ) = sum(abs( tmp(:) ).^2) / 2;          % Divide by 2 Rx ant.
                end
                sinr(ir) = 10*log10( max(rsrp_p0)/(sum(rsrp_p0)-max(rsrp_p0)) );
            end
            iFs = select_fequency(iF); tX = (tx+0.12*iF)*wx+xm;
            ln(end+1) = plot( bins, 100*qf.acdf(sinr,bins),['-',line_col{iFs}],'Linewidth',2);
            plot( sinr38900(iFs,:,il,ic), 5:5:95,['+--',line_col{iFs}],'Linewidth',1 )
            text( tX,ty,   num2str(median(sinr)             ,'%1.1f'),'Color',line_col{iFs});
            text( tX,ty-4, num2str(sinr38900(iFs,10,il,ic) ,'%1.1f'),'Color',line_col{iFs});
        end
        hold off; grid on; box on; set(gca,'YTick',0:10:100);
        set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
        xlabel('Wideband SINR (dB)'); title([ l(1,il).name,' - Config ',num2str( ic ) ] );
        if ic==1; ylabel('CDF [%]'); end;
        legend(ln,legend_names(select_fequency),'Location', 'SouthEast');
        text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
    end
end
```







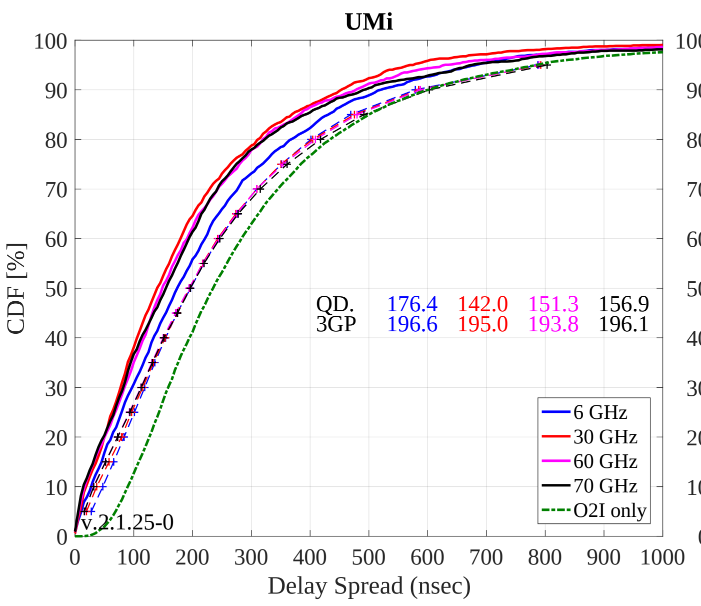


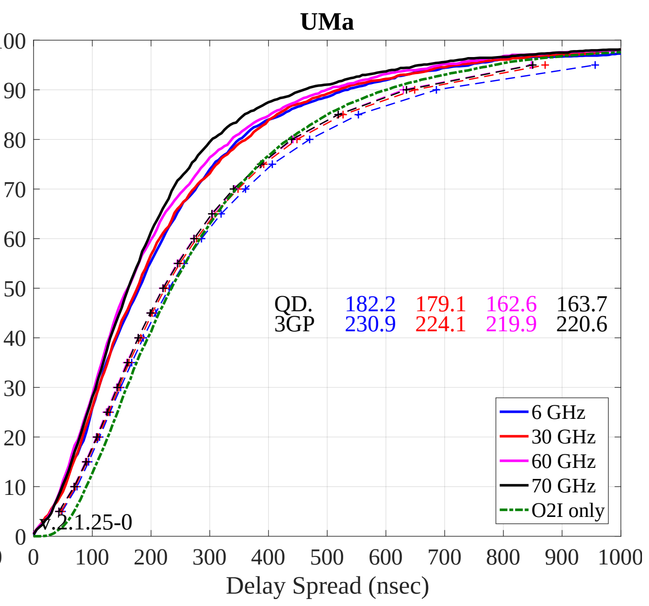


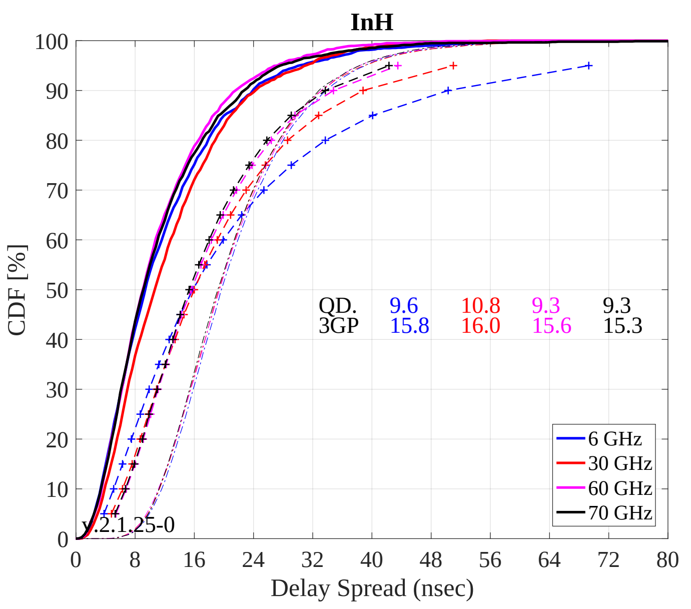
Delay Spread The following plots show the delay spread without antenna patterns for the serving BS, i.e. only the multi-path components are generated by the SSF model, but the paths are not weights by the antenna patterns. For the UMi and UMa scenarios, 80% of the users are indoors. Hence, the results are dominated by the O2I parameters, which are not frequency dependent and are identical for LOS or NLOS propagation of the outdoor link. The green curve therefore shows the O2I distributions of the DS. One can see that the results for UMi and UMa are very similar.

```
legend_ref = {'O2I only','O2I only','InH LOS'};

ref_O2I = 10.^( randn(1,10000)*0.32-6.62 )*1e9;
mu      = (-7.692  -0.01 *log10(1+s.center_frequency'/1e9));
ref_InH = 10.^( 0.18*randn(no_freq,10000) + mu * ones(1,10000) )*1e9;
for il = select_scenario                                              % Scenario
    pg_eff = []; ds = []; ds_tmp = [];
    if il == 1 || il == 3; figure('Position',[ 50 , 550 , 1400 , 640]); end; tx = 0.41; ty = 47;
    if il == 1; axes('position',[0.06 0.12 0.44 0.81]); hold on; xm = 0; wx = 1000; end;
    if il == 2; axes('position',[0.54 0.12 0.44 0.81]); hold on; xm = 0; wx = 1000; end;
    if il == 3; axes('position',[0.30 0.12 0.44 0.81]); hold on; xm = 0; wx = 80; end;
    text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
    ln = []; bins = (-0.1:0.005:1.1)*wx+xm;
    for iF = 1 : no_freq                                              % Frequency
        for ir = 1 : no_rx                                            % Calc. coupling loss
            for it = 1 : size(c{1,il},2)
                pg_eff( it ) = sum(abs(c{1,il}(ir,it,iF).coeff(:)).^2) / 8;
                ds_tmp( it ) = c{1,il}(ir,it,iF).par.ds_cb;
            end
            [~,ii] = max( pg_eff ); ds(ir) = ds_tmp(ii)*1e9;
        end
        iFs = select_fequency(iF); tX = (tx+0.12*iF)*wx+xm;
        ln(end+1) = plot( bins, 100*qf.acdf(ds,bins),['-',line_col{iFs}],'Linewidth',2);
        plot( ds38900(iFs,:,il), 5:5:95,['+--',line_col{iFs}],'Linewidth',1 )
        text( tX,ty,   num2str(median(ds)          ,'%1.1f'),'Color',line_col{iFs});
        text( tX,ty-4, num2str(ds38900(iFs,10,il) ,'%1.1f'),'Color',line_col{iFs});
        if il==3;plot(bins,100*qf.acdf(ref_InH(iF,:),bins),'-.','Color',line_col{iFs});end;
    end
    if il<3; ln(end+1)=plot(bins,100*qf.acdf(ref_O2I,bins),'-.','Color',[0 .5 0],'Linewidth',2);end;
    hold off; grid on; box on; set(gca,'YTick',0:10:100);
    set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
    xlabel('Delay Spread (nsec)'); title([ l(1,il).name ] );
    if il==1 || il==3; ylabel('CDF [%]'); end;
    legend(ln,{legend_names{select_fequency},legend_ref{il}},'Location', 'SouthEast');
    text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
end
```

```
Warning: Ignoring extra legend entries.
```





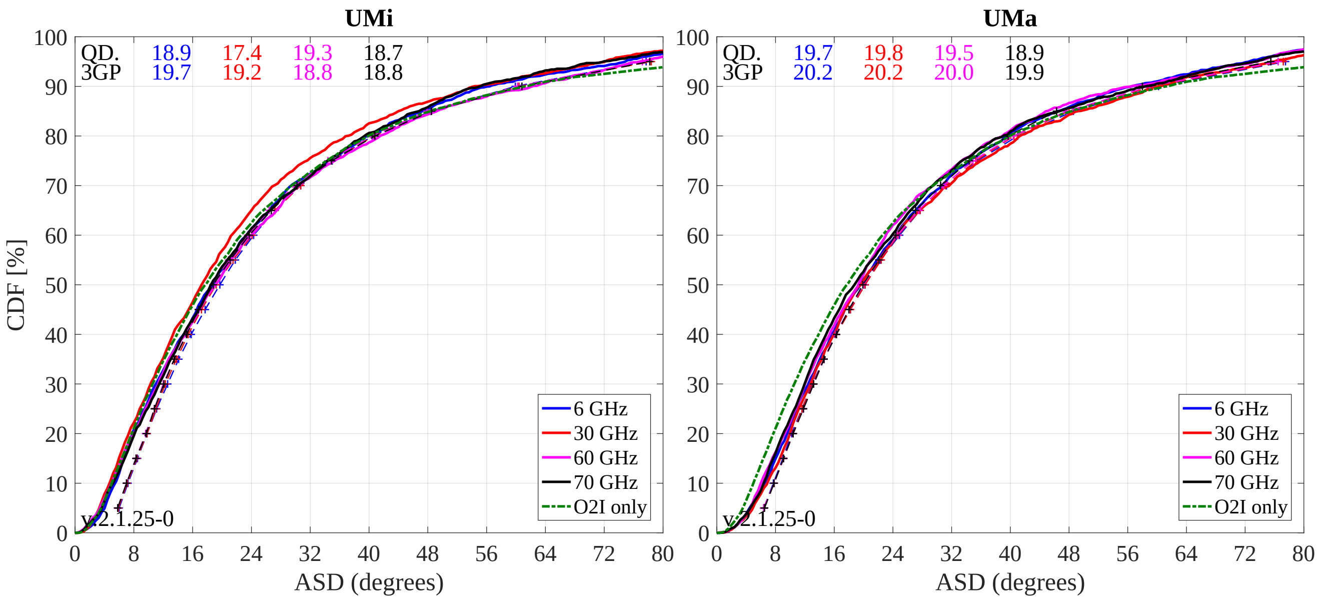


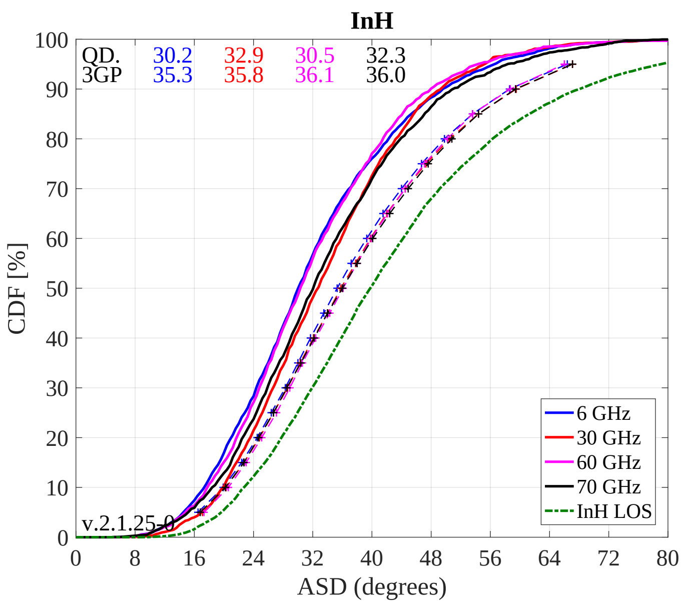
For the InH case, parameters changed in TR 38.901. The original parameterization in 3GPP TR 38.900 V14.0.0 included some significant dependence of the STD of the DS on the carrier frequency. This was removed in TR 38.901. In addition, due to the open office LOS probabilities and the UE attachment to the strongest BS, there are 98% of the users in LOS conditions. Hence, the results heavily depend on the LOS DS, which is shown for the four frequencies are dashed-dotted thin lines. One can observe, that the DS values for the serving BS are always smaller compared to the expected values from the LOS distributions. This comes from the negative correlation of the DS with the SF (-0.8 for InH-LOS). If the link has a high SF, it also has a low DS. However, if the SF is high, the BS gets selected for the UE attachment. As a result, DS values for the serving BS are always smaller compared to the average LOS-DS from all BS.

Azimuth Angle Spread of Departure The next plot shows the ASD. The same assumptions as for the DS apply, i.e. no antenna patterns, UE attachment to the strongest BS, O2I-dominance for UMa and UMi and LOS dominance for InH. Results agree well for UMa and UMi.

```
ref_O2I = 10.^( randn(1,10000)*0.42 +1.25 );
ref_InH = 10.^( randn(1,10000)*0.18 +1.60 );
```

```
for il = select_scenario                                                  % Scenario
    pg_eff = []; asd = []; asd_tmp = [];
    if il == 1 || il == 3; figure('Position',[ 50 , 550 , 1400 , 640]); end; tx = 0.01; ty = 97;
    if il == 1; axes('position',[0.06 0.12 0.44 0.81]); hold on; xm = 0; wx = 80; end;
    if il == 2; axes('position',[0.54 0.12 0.44 0.81]); hold on; xm = 0; wx = 80; end;
    if il == 3; axes('position',[0.30 0.12 0.44 0.81]); hold on; xm = 0; wx = 80; end;
    text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
    ln = []; bins = (-0.1:0.005:1.1)*wx+xm;
    for iF = 1 : no_freq                                                  % Frequency
        for ir = 1 : no_rx                                                % Calc. coupling loss
            for it = 1 : size(c{1,il},2)
                pg_eff( it ) = sum(abs(c{1,il}(ir,it,iF).coeff(:)).^2) / 8;
                asd_tmp( it ) = c{1,il}(ir,it,iF).par.asD_cb;
            end
            [~,ii] = max( pg_eff ); asd(ir) = asd_tmp(ii);
        end
        iFs = select_fequency(iF); tX = (tx+0.12*iF)*wx+xm;
        ln(end+1) = plot( bins, 100*qf.acdf(asd,bins),['-',line_col{iFs}],'Linewidth',2);
        plot( asd38900(iFs,:,il), 5:5:95,['+--',line_col{iFs}],'Linewidth',1 )
        text( tX,ty,   num2str(median(asd)          ,'%1.1f'),'Color',line_col{iFs});
        text( tX,ty-4, num2str(asd38900(iFs,10,il)  ,'%1.1f'),'Color',line_col{iFs});
    end
    if il<3; ln(end+1)=plot(bins,100*qf.acdf(ref_O2I,bins),'-.','Color',[0 .5 0],'Linewidth',2);end;
    if il==3;ln(end+1)=plot(bins,100*qf.acdf(ref_InH,bins),'-.','Color',[0 .5 0],'Linewidth',2);end;
    hold off; grid on; box on; set(gca,'YTick',0:10:100);
    set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
    xlabel('ASD (degrees)'); title([ l(1,il).name ] );
    if il==1 || il==3; ylabel('CDF [%]'); end;
    legend(ln,{legend_names{select_fequency},legend_ref{il}},'Location', 'SouthEast');
    text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
end
```

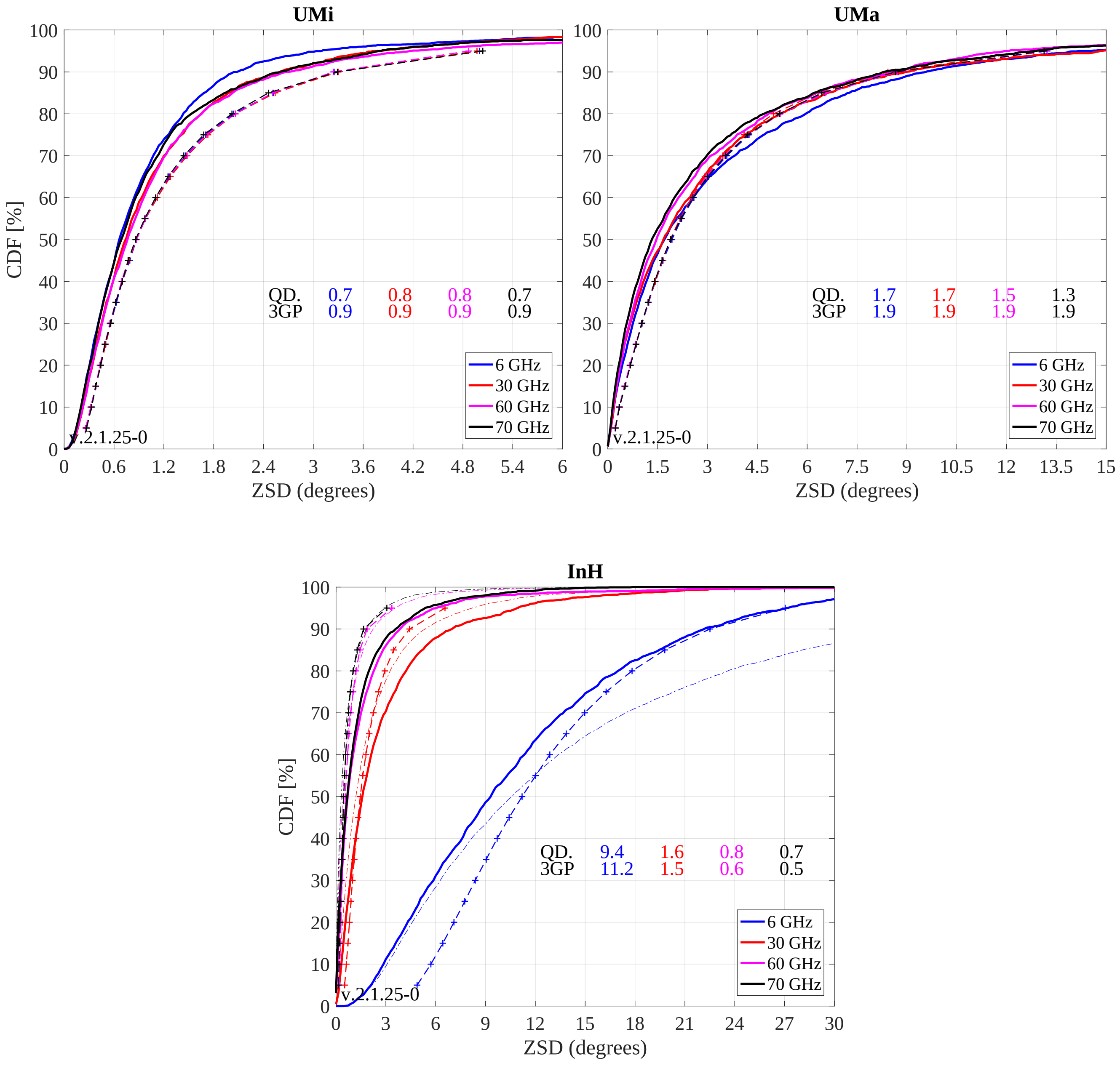




For the InH results, the green curve shows the expected results when only taking the parameters for the LOS-ASD into account. One can see that the results obtained from the model are much lower. This can have two reasons. First, the ASD is negatively correlated with the SF (-0.4). Hence, BSs with a high SF are more likely to become the serving BSs which leads to decreased ASD values for the serving link. Second, the maximum achievable angular spread depends on the KF. The average KF in the indoor LOS scenario is 7 dB. In this case, the maximal achievable azimuth spread is around 50 degree. However, the positive correlation between SF and KF (+0.5) leads to increased KF values for the serving link. As a result, the median ASD for the serving link gets reduced to roughly 30 degree compared to the 40 degree that would be expected from the InH-LOS parameters.

Elevation / Zenith Angle Spread of Departure The next plot shows the ESD / ZSD. The same assumptions as for the DS and ASD apply, i.e. no antenna patterns, UE attachment to the strongest BS, O2I-dominance for UMa and UMi and LOS dominance for InH. Results agree well for UMi and UMa as well as for the higher Frequencies for InH.

```
mu  = (2.228 - 1.43 *log10(1+s.center_frequency'/1e9));
sig = (0.3   + 0.13 *log10(1+s.center_frequency'/1e9));
ref_InH = 10.^( sig * randn(1,10000) + mu * ones(1,10000) );
for il = select_scenario                                             % Scenario
    pg_eff = []; esd = []; esd_tmp = [];
    if il == 1 || il == 3; figure('Position',[ 50 , 550 , 1400 , 640]); end; tx = 0.41; ty = 37;
    if il == 1; axes('position',[0.06 0.12 0.44 0.81]); hold on; xm = 0; wx = 6; end;
    if il == 2; axes('position',[0.54 0.12 0.44 0.81]); hold on; xm = 0; wx = 15; end;
    if il == 3; axes('position',[0.30 0.12 0.44 0.81]); hold on; xm = 0; wx = 30; end;
    text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
    ln = []; bins = (-0.1:0.005:1.1)*wx+xm;
    for iF = 1 : no_freq                                             % Frequency
        for ir = 1 : no_rx                                           % Calc. coupling loss
            for it = 1 : size(c{1,il},2)
                pg_eff( it ) = sum(abs(c{1,il}(ir,it,iF).coeff(:)).^2) / 8;
                esd_tmp( it ) = c{1,il}(ir,it,iF).par.esD_cb;
            end
            [~,ii] = max( pg_eff ); esd(ir) = esd_tmp(ii);
        end
        iFs = select_fequency(iF); tX = (tx+0.12*iF)*wx+xm;
        ln(end+1) = plot( bins, 100*qf.acdf(esd,bins),['-',line_col{iFs}],'Linewidth',2);
        plot( zsd38900(iFs,:,il), 5:5:95,['+--',line_col{iFs}],'Linewidth',1 )
        text( tX,ty,   num2str(median(esd)        ,'%1.1f'),'Color',line_col{iFs});
        text( tX,ty-4, num2str(zsd38900(iFs,10,il) ,'%1.1f'),'Color',line_col{iFs});
        if il==3;plot(bins,100*qf.acdf(ref_InH(iF,:),bins),'-.','Color',line_col{iFs});end;
    end
    hold off; grid on; box on; set(gca,'YTick',0:10:100);
    set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
    xlabel('ZSD (degrees)'); title([ l(1,il).name ] );
    if il==1 || il==3; ylabel('CDF [%]'); end;
    legend(ln,legend_names(select_fequency),'Location', 'SouthEast');
    text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
end
```



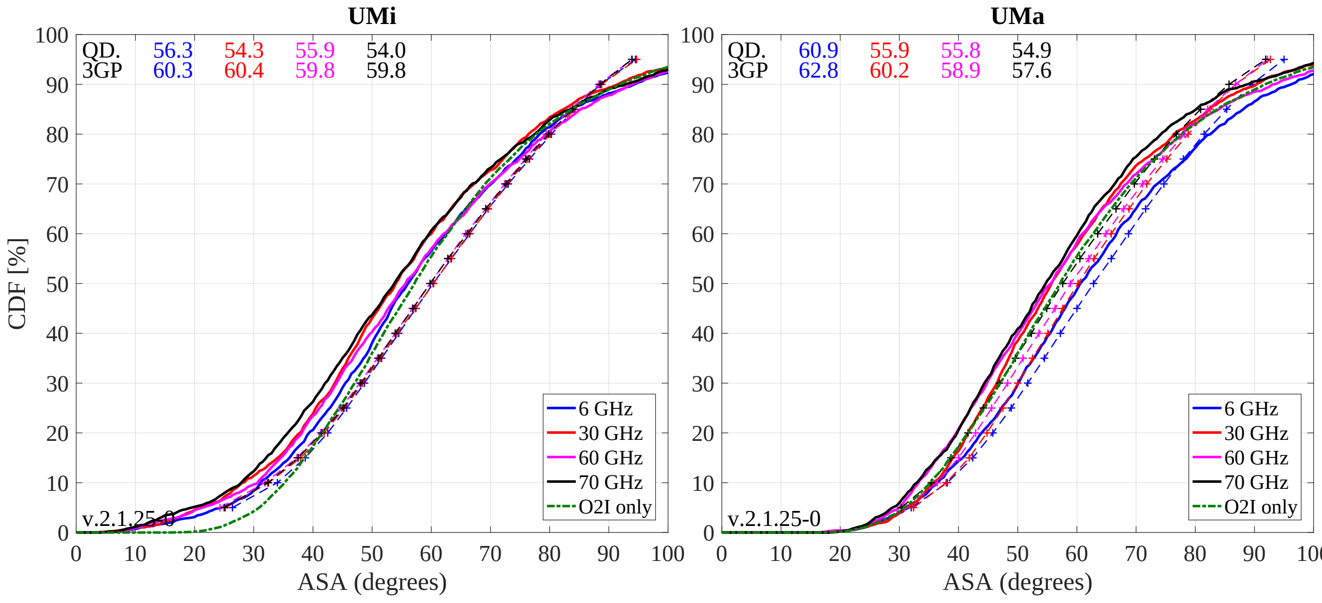
InH at 6 GHz shows some differences for the same reason, the ASD was smaller then expected. The ZSD at lower frequencies can have values above 30 degrees. However, with a KF of 7 dB, the maximum achievable ZSD is around 30 degree. Due to the correlation between SF and KF, the serving link gets even higher KF values and, as a consequence, lower angular spreads.

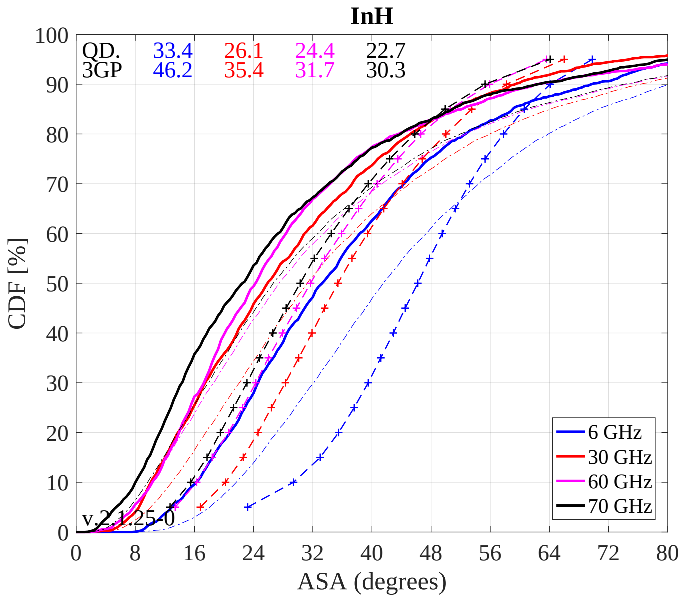
Azimuth Angle Spread of Arrival The next plot shows the ASA. The same assumptions as for the DS apply, i.e. no antenna patterns, UE attachment to the strongest BS, O2I-dominance for UMa and UMi and LOS dominance for InH. Results agree well for UMa and UMi. For InH, the ASA parameters changed from TR 38.900 to 38.901. Hence, different results are obtained at the output of the model compared to the

3GPP calibration reference. At 6 GHz, where the largest ASAs are achieved for InH-LOS, the upper limit for the angle spread is reached due to the correlations of ASA vs. SF (-0.5) and SF vs. KF (+0.5).

```
ref_O2I  = 10.^( randn(1,10000)*0.16 +1.76 );
mu   = (1.781 - 0.19 *log10(1+s.center_frequency'/1e9));
sig  = (0.119 + 0.12 *log10(1+s.center_frequency'/1e9));
ref_InH = 10.^( sig * randn(1,10000) + mu * ones(1,10000) );
for il = select_scenario                                            % Scenario
    pg_eff = []; asa = []; asa_tmp = [];
    if il == 1 || il == 3; figure('Position',[ 50 , 550 , 1400 , 640]); end; tx = 0.01; ty = 97;
    if il == 1; axes('position',[0.06 0.12 0.44 0.81]); hold on; xm = 0; wx = 100; end;
    if il == 2; axes('position',[0.54 0.12 0.44 0.81]); hold on; xm = 0; wx = 100; end;
    if il == 3; axes('position',[0.30 0.12 0.44 0.81]); hold on; xm = 0; wx = 80; end;
    text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
    ln = []; bins = (-0.1:0.005:1.1)*wx+xm;
    for iF = 1 : no_freq                                            % Frequency
        for ir = 1 : no_rx                                          % Calc. coupling loss
            for it = 1 : size(c{1,il},2)
                pg_eff( it ) = sum(abs(c{1,il}(ir,it,iF).coeff(:)).^2) / 8;
                asa_tmp( it ) = c{1,il}(ir,it,iF).par.asA_parset;
            end
            [~,ii] = max( pg_eff ); asa(ir) = asa_tmp(ii);
        end
        iFs = select_fequency(iF); tX = (tx+0.12*iF)*wx+xm;
        ln(end+1) = plot( bins, 100*qf.acdf(asa,bins),['-',line_col{iFs}],'Linewidth',2);
        plot( asa38900(iFs,:,il), 5:5:95,['+--',line_col{iFs}],'Linewidth',1 )
        text( tX,ty,   num2str(median(asa)          ,'%1.1f'),'Color',line_col{iFs});
        text( tX,ty-4, num2str(asa38900(iFs,10,il)  ,'%1.1f'),'Color',line_col{iFs});
        if il==3;plot(bins,100*qf.acdf(ref_InH(iF,:),bins),'-.','Color',line_col{iFs});end;
    end
    if il<3; ln(end+1)=plot(bins,100*qf.acdf(ref_O2I,bins),'-.','Color',[0 .5 0],'Linewidth',2);end;
    hold off; grid on; box on; set(gca,'YTick',0:10:100);
    set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
    xlabel('ASA (degrees)'); title([ l(1,il).name ] );
    if il==1 || il==3; ylabel('CDF [%]'); end;
    legend(ln,{legend_names{select_fequency},legend_ref{il}},'Location', 'SouthEast');
    text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
end
```

```
Warning: Ignoring extra legend entries.
```

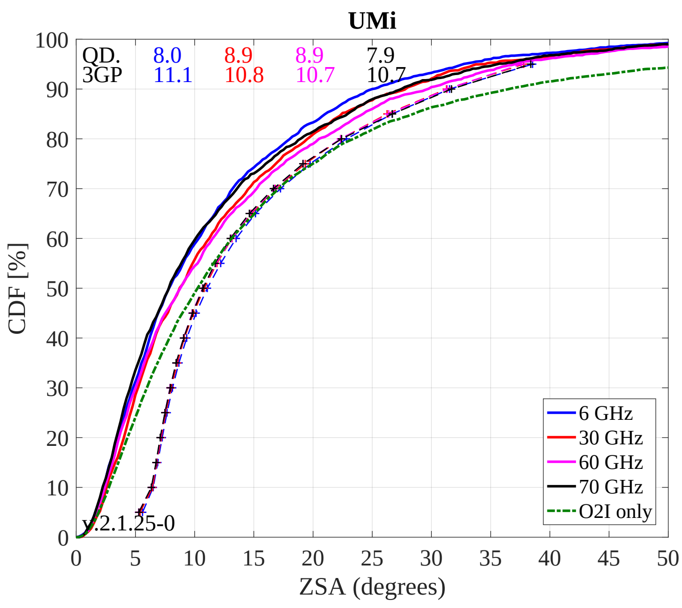


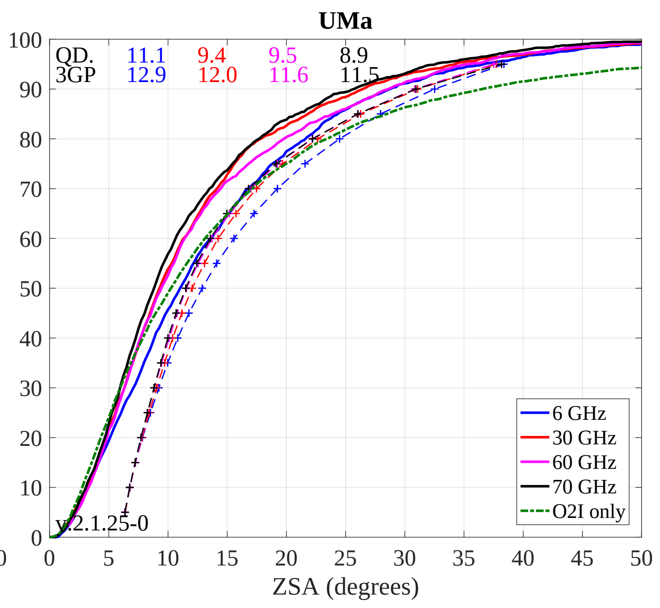


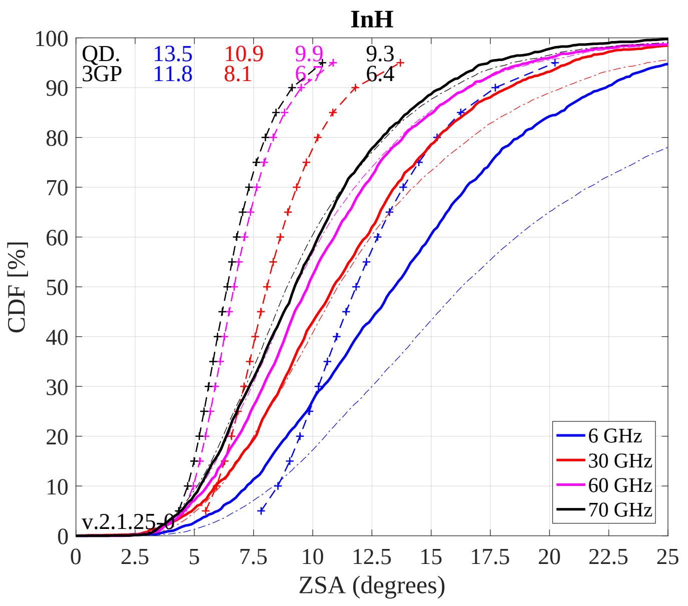
Elevation / Zenith Angle Spread of Arrival The next plot shows the ESD / ZSD results. Again, UMi and UMa results agree well since mostly, O2I parameters apply (green curve). For InH, the 3GPP parameters changed from TR 38.900 to 38.901. Hence, different results are obtained at the output of the model compared to the 3GPP calibration reference.

```
ref_O2I = 10.^( randn(1,10000)*0.43 +1.01 );
mu  = (1.44  - 0.26 *log10(1+s.center_frequency'/1e9));
sig = (0.264 - 0.04 *log10(1+s.center_frequency'/1e9));
ref_InH = 10.^( sig * randn(1,10000) + mu * ones(1,10000) );
for il = select_scenario                                        % Scenario
    pg_eff = []; esa = []; esa_tmp = [];
    if il == 1 || il == 3; figure('Position',[ 50 , 550 , 1400 , 640]); end; tx = 0.01; ty = 97;
    if il == 1; axes('position',[0.06 0.12 0.44 0.81]); hold on; xm = 0; wx = 50; end;
    if il == 2; axes('position',[0.54 0.12 0.44 0.81]); hold on; xm = 0; wx = 50; end;
    if il == 3; axes('position',[0.30 0.12 0.44 0.81]); hold on; xm = 0; wx = 25; end;
    text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
    ln = []; bins = (-0.1:0.005:1.1)*wx+xm;
    for iF = 1 : no_freq                                        % Frequency
        for ir = 1 : no_rx                                      % Calc. coupling loss
            for it = 1 : size(c{1,il},2)
                pg_eff( it ) = sum(abs(c{1,il}(ir,it,iF).coeff(:)).^2) / 8;
                esa_tmp( it ) = c{1,il}(ir,it,iF).par.esA_cb;
            end
            [~,ii] = max( pg_eff ); esa(ir) = esa_tmp(ii);
        end
        iFs = select_fequency(iF); tX = (tx+0.12*iF)*wx+xm;
        ln(end+1) = plot( bins, 100*qf.acdf(esa,bins),['-',line_col{iFs}],'Linewidth',2);
        plot( zsa38900(iFs,:,il), 5:5:95,['+--',line_col{iFs}],'Linewidth',1 )
        text( tX,ty,   num2str(median(esa)          ,'%1.1f'),'Color',line_col{iFs});
        text( tX,ty-4, num2str(zsa38900(iFs,10,il)  ,'%1.1f'),'Color',line_col{iFs});
        if il==3;plot(bins,100*qf.acdf(ref_InH(iF,:),bins),'-.','Color',line_col{iFs});end;
    end
    if il<3; ln(end+1)=plot(bins,100*qf.acdf(ref_O2I,bins),'-.','Color',[0 .5 0],'Linewidth',2);end;
    hold off; grid on; box on; set(gca,'YTick',0:10:100);
    set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
    xlabel('ZSA (degrees)'); title([ l(1,il).name ] );
    if il==1 || il==3; ylabel('CDF [%]'); end;
     legend(ln,{legend_names{select_fequency},legend_ref{il}},'Location', 'SouthEast');
    text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
end
```

```
Warning: Ignoring extra legend entries.
```







First Singular Value (Configuration 2) The singular values of a MIMO channel matrix describe how many parallel spatial data streams can be transmitted to one user and what the individual capacity of each streams is. The antenna configuration results in a 2x4 MIMO channel. Hence, the channel has two singular values and supports at most two streams. The singular values are calculated as follows:

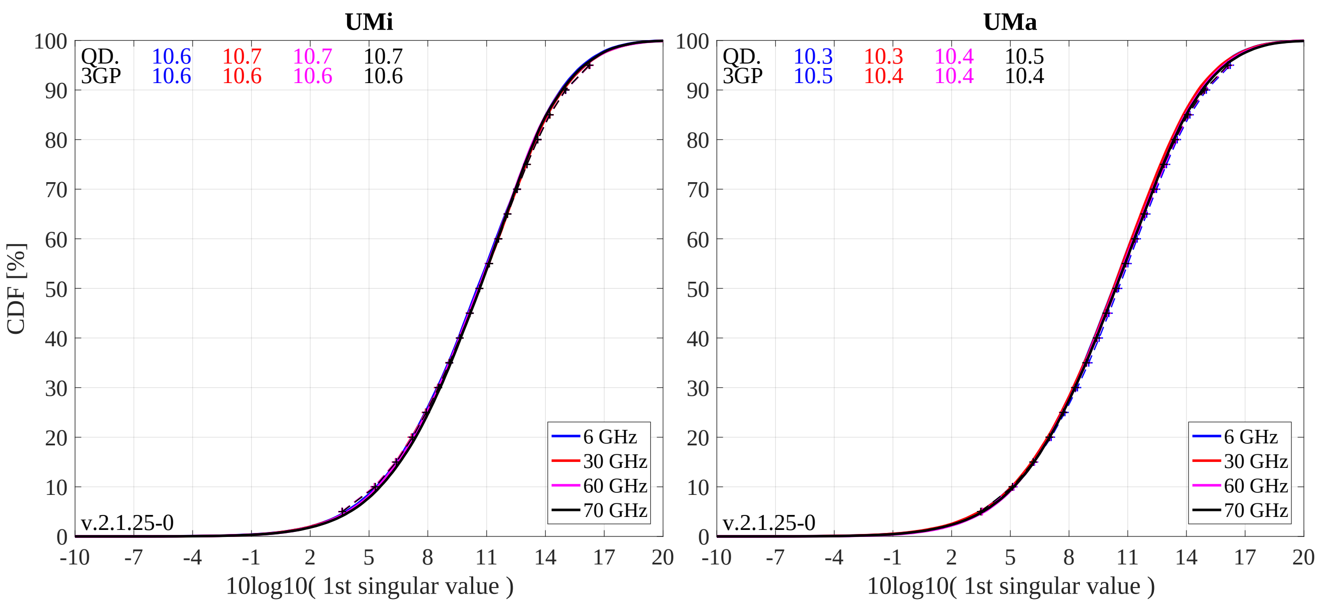
- The results are reported for the channel matrix of the serving BS.
- The calculations are done in the frequency domain. The bandwidth is set to 20 MHz (at 6 GHz) or 100 MHz (above 6 GHz), which is further split into 50 resource blocks (RBs).
- The singular values are reported for channels without path-gain, but with antenna patterns included.
- The singular values are calculated for each RB by an Eigenvalue decomposition of the receive covariance matrix as

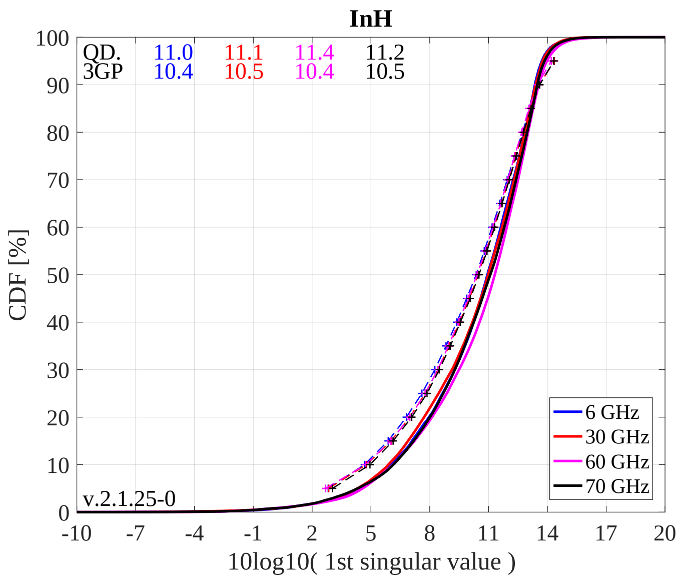
$$s_{1,2} = \frac{1}{n_{RB}} \cdot \text{eig}\left(\sum_{n=1}^{n_{RB}} \mathbf{H}_n \mathbf{H}_n^H \right)$$

- Results are presented in logarithmic scale, i.e. as $10 \cdot \log_{10}(s_{1,2})$.

Results for the first singular value agree perfectly for UMi and UMa. Only minor differences can be seen for InH.

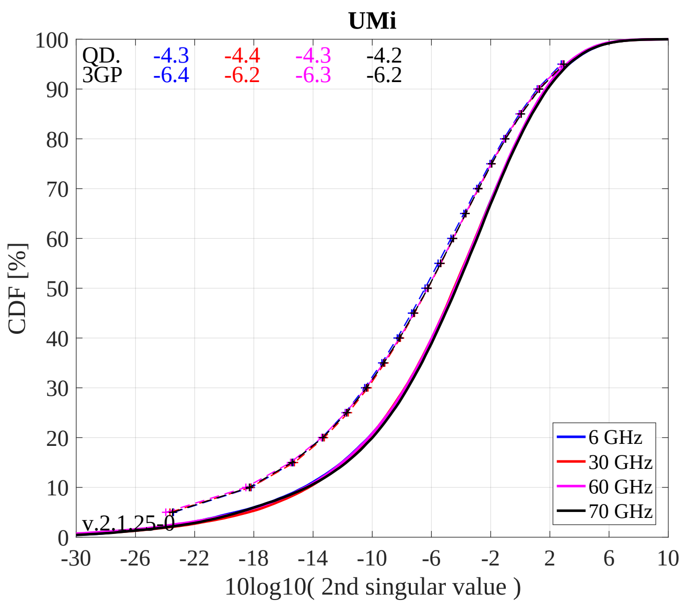
```
clear sv                                                          % Calculate singular values
BW = [20,100,100,100]*1e6;                                        % Bandwidth for each
for il = select_scenario
    sv{il} = zeros( 2,50,no_rx,4 );
    pg_eff = [];
    for iF = 1 : no_freq
        for ir = 1 : no_rx
            for it = 1 : size(c{2,il},2)
                pg_eff( it ) = sum(abs(c{2,il}(ir,it,iF).coeff(:)).^2) / 8;
            end
            [~,it] = max( pg_eff );                               % Select serving BS
            H  = c{2,il}(ir,it,iF).fr( BW(iF), 50 );
            pg = c{2,il}(ir,it,iF).par.pg_parset;
            H = H ./ sqrt(10.^(0.1*pg));                          % Normalize channel matrix
            for m = 1:size(H,3)
                sv{il}(:,m,ir,iF) = svd(H(:,:,m)).^2;
            end   % NOTE: eig( H(:,:,m)*H(:,:,m)' ) == svd(H(:,:,m)).^2
        end
    end
end

for il = select_scenario                                                  % Scenario
    pg_eff = []; esa = []; esa_tmp = [];
    if il == 1 || il == 3; figure('Position',[ 50 , 550 , 1400 , 640]); end; tx = 0.01; ty = 97;
    if il == 1; axes('position',[0.06 0.12 0.44 0.81]); hold on; xm = -10; wx = 30; end;
    if il == 2; axes('position',[0.54 0.12 0.44 0.81]); hold on; xm = -10; wx = 30; end;
    if il == 3; axes('position',[0.30 0.12 0.44 0.81]); hold on; xm = -10; wx = 30; end;
    text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
    ln = []; bins = (-0.1:0.005:1.1)*wx+xm;
    for iF = 1 : no_freq                                                  % Frequency
        sv_max = 10*log10( reshape(sv{il}(1,:,:,iF),[],1) );
        iFs = select_fequency(iF); tX = (tx+0.12*iF)*wx+xm;
        ln(end+1) = plot( bins, 100*qf.acdf(sv_max,bins),['-',line_col{iFs}],'Linewidth',2);
        plot( sv1_38900(iFs,:,il), 5:5:95,['+--',line_col{iFs}],'Linewidth',1 )
        text( tX,ty,   num2str(median(sv_max)      ,'%1.1f'),'Color',line_col{iFs});
        text( tX,ty-4, num2str(sv1_38900(iFs,10,il) ,'%1.1f'),'Color',line_col{iFs});
    end
    hold off; grid on; box on; set(gca,'YTick',0:10:100);
    set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
    xlabel('10log10( 1st singular value )'); title([ l(1,il).name ] );
    if il==1 || il==3; ylabel('CDF [%]'); end;
    legend(ln,legend_names(select_fequency),'Location', 'SouthEast');
    text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
end
```

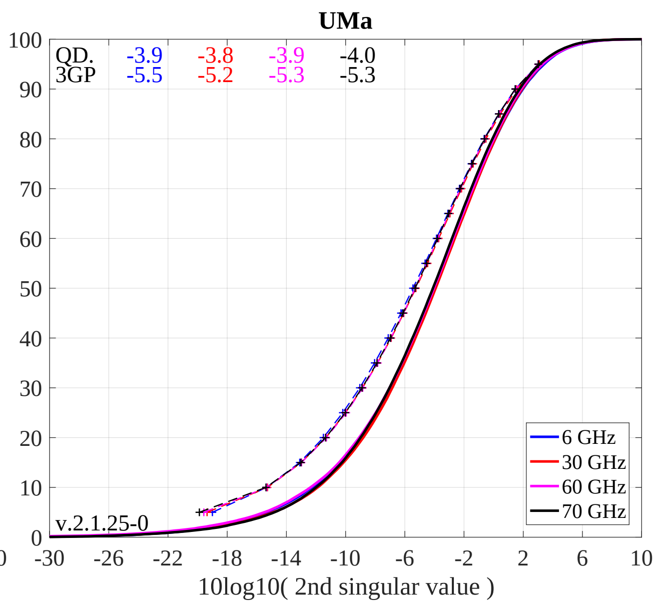


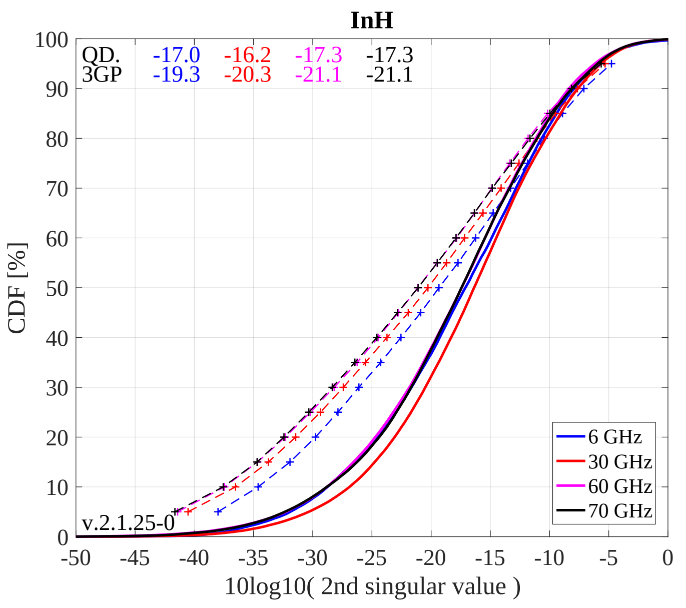


Second Singular Value (Configuration 2) Results are slightly larger for UMi and UMa indicating a slightly higher channel capacity as reported by the average 3GPP results. However, the results presented here are still well within the range of the results reported by different partners in R1-165975. For InH, the larger differences are probably due to the changed parameterization TR 38.901.

```
for il = select_scenario                                                    % Scenario
    pg_eff = []; esa = []; esa_tmp = [];
    if il == 1 || il == 3; figure('Position',[ 50 , 550 , 1400 , 640]); end; tx = 0.01; ty = 97;
    if il == 1; axes('position',[0.06 0.12 0.44 0.81]); hold on; xm = -30; wx = 40; end;
    if il == 2; axes('position',[0.54 0.12 0.44 0.81]); hold on; xm = -30; wx = 40; end;
    if il == 3; axes('position',[0.30 0.12 0.44 0.81]); hold on; xm = -50; wx = 50; end;
    text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
    ln = []; bins = (-0.1:0.005:1.1)*wx+xm;
    for iF = 1 : no_freq                                                    % Frequency
        sv_min = 10*log10( reshape(sv{il}(2,:,:,iF),[],1) );
        iFs = select_fequency(iF); tX = (tx+0.12*iF)*wx+xm;
        ln(end+1) = plot( bins, 100*qf.acdf(sv_min,bins),['-',line_col{iFs}],'Linewidth',2);
        plot( sv2_38900(iFs,:,il), 5:5:95,['+--',line_col{iFs}],'Linewidth',1 )
        text( tX,ty,   num2str(median(sv_min)      ,'%1.1f'),'Color',line_col{iFs});
        text( tX,ty-4, num2str(sv2_38900(iFs,10,il) ,'%1.1f'),'Color',line_col{iFs});
    end
    hold off; grid on; box on; set(gca,'YTick',0:10:100);
    set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
    xlabel('10log10( 2nd singular value )'); title([ l(1,il).name ] );
    if il==1 || il==3; ylabel('CDF [%]'); end;
    legend(ln,legend_names(select_fequency),'Location', 'SouthEast');
    text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
end
```





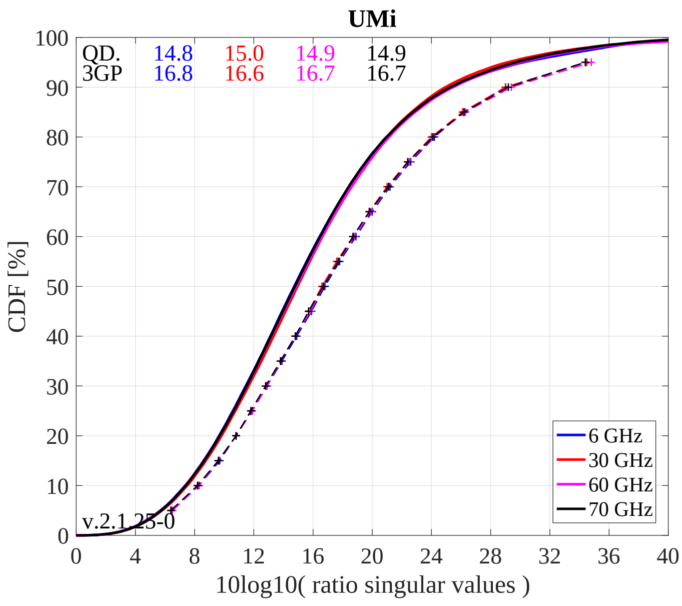


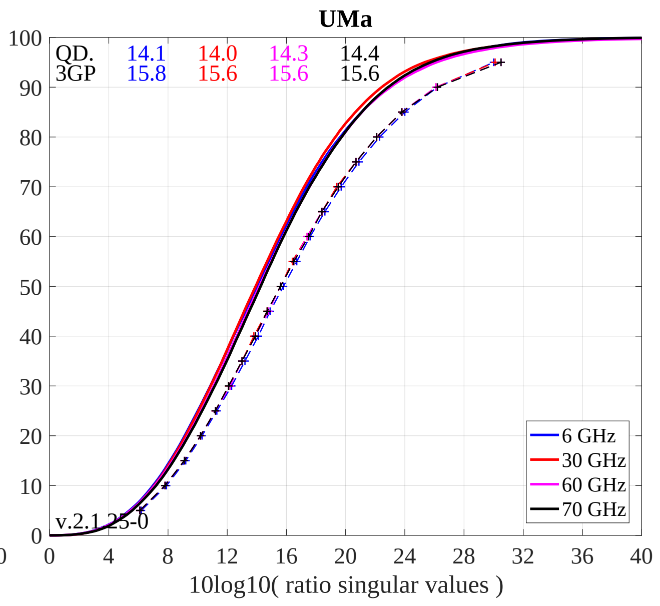
Ratio of Singular Values (Configuration 2) Probably a more important measure than the singular values themselves is the ratio between the singular values, which is calculated as

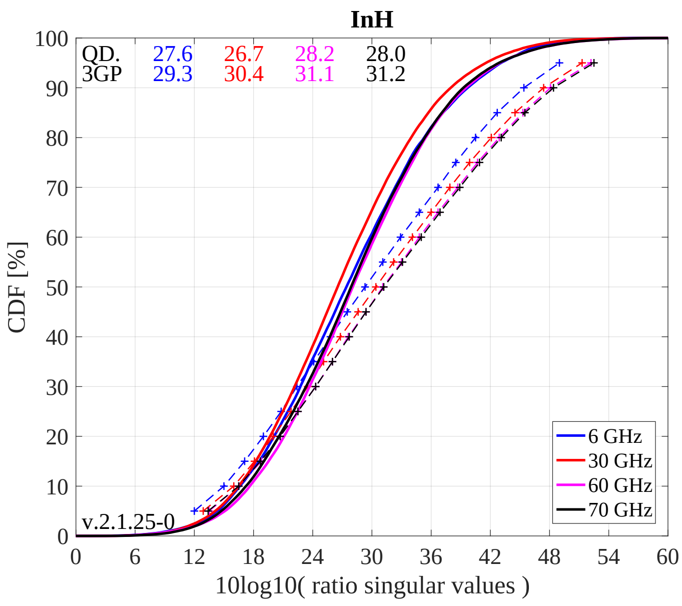
$$SR = 10 \cdot \log_{10}\left(\frac{s_1}{s_2}\right)$$

This measure is closely linked to the condition number of the channel matrix $C = \sqrt{\frac{s_1}{s_2}}$. The larger this number is, the more difficult it is to invert the matrix **H**. However, inverting this matrix is required in order to separate the two data streams at the receiver. Due to the larger second SV our results are better (lower values are better) that the 3GPP baseline for UMa and UMi. The InH cannot be discussed properly due to the changed parameterization TR 38.901.

```
for il = select_scenario                                                % Scenario
    pg_eff = []; esa = []; esa_tmp = [];
    if il == 1 || il == 3; figure('Position',[ 50 , 550 , 1400 , 640]); end; tx = 0.01; ty = 97;
    if il == 1; axes('position',[0.06 0.12 0.44 0.81]); hold on; xm = 0; wx = 40; end;
    if il == 2; axes('position',[0.54 0.12 0.44 0.81]); hold on; xm = 0; wx = 40; end;
    if il == 3; axes('position',[0.30 0.12 0.44 0.81]); hold on; xm = 0; wx = 60; end;
    text( tx*wx+xm,ty,'QD.'); text( tx*wx+xm,ty-4,'3GP');
    ln = []; bins = (-0.1:0.005:1.1)*wx+xm;
    for iF = 1 : no_freq                                              % Frequency
        sv_rat = 10*log10( reshape( sv{il}(1,:,:,iF)./sv{il}(2,:,:,iF) ,[],1) );
        iFs = select_fequency(iF); tX = (tx+0.12*iF)*wx+xm;
        ln(end+1) = plot( bins, 100*qf.acdf(sv_rat,bins),['-',line_col{iFs}],'Linewidth',2);
        plot( svR_38900(iFs,:,il), 5:5:95,['+--',line_col{iFs}],'Linewidth',1 )
        text( tX,ty,   num2str(median(sv_rat)       ,'%1.1f'),'Color',line_col{iFs});
        text( tX,ty-4, num2str(svR_38900(iFs,10,il) ,'%1.1f'),'Color',line_col{iFs});
    end
    hold off; grid on; box on; set(gca,'YTick',0:10:100);
    set(gca,'XTick',xm : wx/10 : xm+wx); axis([xm xm+wx 0 100]);
    xlabel('10log10( ratio singular values )'); title([ l(1,il).name ] );
    if il==1 || il==3; ylabel('CDF [%]'); end;
    legend(ln,legend_names(select_fequency),'Location', 'SouthEast');
    text( 0.01*wx+xm, 3, ['v.',qd_simulation_parameters.version] );
end
```


UMi
QD. 14.8 15.0 14.9 14.9
3GP 16.8 16.6 16.7 16.7
CDF [%]
10log10(ratio singular values)
v.2.1.25-0
6 GHz
30 GHz
60 GHz
70 GHz


UMa
QD. 14.1 14.0 14.3 14.4
3GP 15.8 15.6 15.6 15.6
10log10(ratio singular values)
v.2.1.25-0
6 GHz
30 GHz
60 GHz
70 GHz


InH
QD. 27.6 26.7 28.2 28.0
3GP 29.3 30.4 31.1 31.2
CDF [%]
10log10(ratio singular values)
v.2.1.25-0
6 GHz
30 GHz
60 GHz
70 GHz